Computational Augmentation of Model Based System Engineering

Supporting Mechatronic System Model Development with AI Technologies

Mechatronics Research Centre

Faculty of Technology

De Montfort University, United Kingdom

This thesis is submitted in partial fulfilment

of the requirements of De Montfort University

for the award of Doctor of Philosophy

December, 2019

By:
Jugraj Singh

Table of Contents

TABLE OF FIGURES

TABLE OF TABLES

DECLARATION

No part of the material described in this thesis has been submitted for the award of any other degree or qualification in this or any other university or college of advanced education.

Jugraj Singh

ABSTRACT

Efforts in applying computational support for automatic design synthesis and configuration generation as well as efforts to support descriptive and computational model development for system design and verification has been approached with semantic formalisation of modelling languages and of generic structural and functional concepts using meta-models. Modelling the system using descriptive models helps the designer to explicitly document dependencies between properties and parameters of system and external entities.

The descriptive models thus produced often do not consider physics based justification for presence and/or absence of relations. It is often the case, the simulation results obtained at later stages requires changing requirements as well as modifying logical (modelling relations between high level functions parameters/properties and parameters/properties of high level entities) and physical architectures (modelling relations between component's parameters and properties) to accommodate those requirements. The current MBSE (Model Based System Engineering) tools have capabilities to verify construction of models according to predefined model formats i.e. meta-models. However, these tools and current research in augmenting capabilities of these tools lacks the focus on evaluating content inside the models i.e. whether the system modelled by models represents a system that can be physically realized.

This work has tried to avail the potential of available AI (Artificial Intelligence) technologies for assisting modelling activities performed for requirement definition and analysis, architecture design and verification phase of system development process by directing designer to tools that can formalise outputs of model development activities. The proposed problem formulation is based on the insight that a system modelled at both conceptual and detailed design level can be represented by logical and mathematical relations between the properties and parameters of internal and external components or functions of the system and domain. Therefore formulation defines concepts used in requirement, logical architecture and physical architecture models using relation between parameters and properties in those models. Concepts, such as operational requirements (or non-functional requirements for particular use case scenario), are defined through the usage of sets and linking value domains of those sets to particular system application domain for which system model is being developed. These relations enables systematic elaboration of requirements into logical and physical architecture models as well as storage and retrieval of existing model knowledge using existing AI tools.

A novel framework has been developed to retrieve existing descriptive structure and function models using logical reasoning as well as to retrieve existing simulation models stored in embedding space of auto-encoder neural network. Beside adopting the concepts of semantic

formalisation and meta-model based descriptive knowledge retrieval it utilises novel application of unsupervised representation learning capability of neural network auto-encoders to store known physically and technologically feasible designs in low dimensional representation that cluster similar designs therefore inducing similarity or distance metric that can be used to retrieve the known design with similar behaviour as new required behaviour. Framework also enable application of generic and domain specific logical constraints (as other works has done before) and introduces new concept of system application domain to ensures that at every stage of the model development leading to conceptual physical design architecture stays inside the physical constraints as per system usage domain. The instantiated meta-model elements which are classified to a system application domain (SAD) are implicitly constraint by system usage context constraints (e.g. parameter value restriction), similarly known simulation models can also be categorised to different SADs.

The proposed framework extends the conventional approach of automated design synthesis which is only based only on decomposition of high level function (summarizing input to output mapping) into basic functions and selecting components to realize those basic functions. "A system is designed with the aim that it can execute its function(s) as per performance requirements of that function(s) in required operational conditions"- By concentrating on this statement it can be seen that conventional approach of functional decomposition and function allocation to known structural components cannot guarantee to yield a working system in required scenarios by ignoring the dependencies between environment or operating conditions and operating modes of prospective designs satisfying high level function.

The results obtained from the implementation of domain specific knowledge representation and retrieval (involving mixture of numerical and logical constraints) as well as the results obtained from implementation of neural network auto-encoder for representation and retrieval of domain specific simulation model demonstrates the viability of these technologies to support the proposed framework.

Overview of knowledge contributed: Chapter 3 partially formalises key system modelling outputs in light of review of MBSE methodologies and then describes a new framework supporting each modelling output's some aspect using existing knowledge through relation inference capability[1] of state of the art AI tools in Chapter 4 and Chapter 5.

[1] Relation deduction and inference can be either to establish descriptive or symbolic relation e.g. predicate "SubSystem *has* components" or a mathematical relation e.g. based on distance metric on a numerical encoding of behaviour signals.

Chapter 1 Introduction

1.1. Introduction

The increase in computation power and development of efficient algorithm has allowed delegation of complex tasks, such as domestic cleaning with robots, image search based on image content, automatic generation of music etc. A similar trend has emerged in system engineering community and in a wide range of system modellers, such as modelling biological process, chemical processing plants, to use ontologies and other semantic web technologies to organise simulation and descriptive models as well as to guide model development by encoding constraints on model elements in ontology. On the other hand, Machine learning especially in field of deep learning has made lot of strides in learning abstract high level concepts e.g. distinguishing faces based on pixel values of an image.

Models has been widely used in engineering for purpose of describing conceptual design topology through use of abstract notations as in electrical schematics and for analysing behaviour of system by defining structural, functional and behavioural constraints on the system. Similarly, system engineers follow a specific Model Based System Engineering (MBSE) methodology to create descriptive models using a modelling language, such as SysML (System Modelling Language). SysML is a standardised language which can be used to specify heterogeneous systems and it extends UML (Unified Modelling Language) by providing additional modelling elements (also called 'concepts') and constructs in diagrams such as parametric diagram, block definition diagram (extension of UML class diagrams) etc. [1], which are used for capturing behaviour, function and structure of system of interest while considering its interaction with entities in surrounding environment for different use case scenarios.

In MBSE, the standard system development phases of requirement definition and analysis, system architecture design, design synthesis, implementation, integration, verification and validation are supported by a centralized system model repository, which is composed of models representing requirements specification, system decomposition in different contexts (also called logical and physical system architecture), use case scenarios, function sequencing or function flows pertaining to particular use case scenario and behaviours in different operational contexts.

These models are often interfaced with external design, analyses and verification tools to exchange information with them to support analysis of different types such as structural optimisation, dynamic behaviour, reliability, failure mode and effect analysis etc. The tools that provides software environment for implementing SysML can also support capabilities like

requirement traceability and completeness checking (e.g. whether all the relations have defined multiplicity or not) because of the semi-formal semantics applied by OCL (object constraint language) constraints [2] on SysML's meta-model elements [3] which defines SysML syntax. But also because of this semi-formal semantics, the models developed in this language are not amenable to any kind of logical reasoning for answering questions related to the content described in the model, e.g. A car model cannot be queried whether the car represented in the model can make a turn at x degrees when travelling at x miles/hour, hence making integration of SysML language directly into an intelligent system (aiding in model based system development) difficult.

However indirect (by translation of SysML models into an intermediate representation) integration can provide capabilities such as automatic descriptive model knowledge organisation (by relating with existing models using some criteria such as class hierarchy) and retrieval. Whereas, potential of deep learning methods to learn bottom up high level feature learning from low level features (i.e. learning shape of nose, ears and other face features at low level) can be utilised in answering questions like "if a simulation model of a car exist which can make a turn at x degrees when travelling at x miles/hour?" without extensive simulation models analysis. Therefore these deep learning methods can help in automatic mapping of descriptive parametric models into existing or modified simulation models that can be used for alternative configuration generation, evaluation and selection during trade-off analysis.

Such question answering capability is only possible if a machine learning algorithm is able to learn relationship between physical system properties (e.g. car's mass, electrical circuit's resistance etc.) and system's parameters at its various I/O conceptual ports (e.g. wheel's linear velocity, current through a load resistor etc.).

1.2. Motivation and Background

A recent survey on MBSE adoption in industry has highlighted the issue of high entry level cost due to training, ambiguous MBSE concept definition and lack of use case examples [4]. [5] has also acknowledged identification of obstacles, other than technical ones, such as training. Similarly, [6] has described problems in usage of standard modelling languages by engineers other than modelling experts.

To tackle these drawbacks domain specific modelling languages has been developed to provide domain specific notations which are easy to understand for non-modelling expert and also provide reasoning and inference support for requirement verification as well as completeness, consistency and conformance checking with respect to defined conceptual data model or meta-model. DSMLs for system engineering purposes are provided by modelling tools like Capella

[6] and Virtual Spacecraft Design (VSD) environment [7] . Other ad-hoc DSMLs have also been developed e.g. to support virtual design and verification of industrial process facility [8], and for supporting integrated product development by use of Semantic Product Modelling Language (SPML) [9].

Most of domain specific tools can also encode domain dependent knowledge in form of design rules e.g. electronics CAD tools uses rules to detect if power tracks are shorted with the ground to highlight signal integrity issues. If such experience and knowledge, which could be general design heuristics or problem specific knowledge, can be encoded in a knowledge base it can be used by intelligent system to advice system engineer on steps available to complete or modify the design.

Knowledge produced during the design process as well as used from previous development projects should be organised automatically, because the inefficient manual handling of complex knowledge during archival and retrieval is cumbersome and prone to error [10] as well as prohibits extensive exploration of design space [11]. The subjective manual decision making based on inconsistent knowledge or information obtained through documents generated from isolated modelling and analyses activities [12] [13] as well as inability to store design rationale related to solution in semantically rigorous form for design decision traceability [14] [13] creates potential for downstream risk e.g. in case of undesired simulation results, design rationale used for design steps taken during system design can be inspected to make a sound modification decision.

SysML, for being a semi-formal language, cannot directly provide such knowledge organisation capability due to its inability to formalise dependencies between the models developed during key system development phases of requirement definition and analysis, system architecture design, design synthesis and verification. The requirement definition and analysis phases provides functional requirements upon which logical or conceptual system architecture is formulated and the design synthesis phase determines the components that will realise those functions. The selection of particular components can induce new component's type specific requirements in the requirement model and thus forcing modifications to other models. Therefore effect of changes initiated by events like discontinuation of parts from supplier, unexpected result of verification activity or change in stakeholder requirement [12] [15], can only be observed in model elements directly affected by change at a local level, such as system component models connected to the modified component and requirements satisfied by that component, but not on overall system function and behaviour.

To infer such cascade of changes and to make them observable at high level system model, tools are required which can reason, not just on format or syntax of the models, but also on the

purpose and content of the models with respect to generic physical laws, constraints of available technology and design methodology while utilising problem specific knowledge from structural, behavioural and functional models of existing systems embodying constraints of available engineering technology. Such capabilities can allow automatic consistent modification of low level information while enabling system engineering to focus on high level model content.

Besides the research in supporting MBSE tools and languages, research in area of design automation has also employed knowledge engineering methods, along with other computational algorithms from artificial intelligence field, with the goal of automatically producing conceptual design [16] and optimising design configuration [15] based on given requirements, knowledge of component's structure/function and knowledge regarding physical constraints acting on the component.

Knowledge engineering methods has also been used for knowledge reuse in case of correct simulation model selection for required purpose [17] [10] and existing solution reuse related to system structure for similar requirements [18]. Expert systems have been developed that follow 'propose and revise' [19] approach for generating design configurations while dealing with partial (due to inexact, uncertain and incomplete) design specification. Semantic web technologies have also been used to develop knowledge base with use of RDF2 (Resource Description Framework) graphs for generating alternative configurations and for verification of interface compatibility [20].

On the other hand, computational design synthesis and configuration (reviewed in 2.2.2.) has used functional decomposition and analogy based methods which uses domain specific and generic knowledge regarding product, its environment, design requirement and design process. The relations between such knowledge elements are generally specified in conceptual schemas. Some of the main approaches used by researchers to aid the system engineer in considering all relevant aspect of the system is based on formalising system models and reusing stored knowledge to elaborate those models and finding solution by using computational techniques such as SAT solvers and simulated annealing. Potential design solutions are generally stored in form of reusable standard structural and constraint models, which are retrieved using standard functions obtained by decomposing functional requirements. Inference of design constraints related to input/output variables given in system requirements is also usually based on knowledge-based methods. Solution, in form of physical component's property or parameter

[2] https://www.w3.org/TR/rdf-concepts/

values, is then selected by solving these design constraint through techniques such as constraint satisfaction.

1.2.1. Research Gap

Various shortcomings in the system design methods and tools (available during start of millennia) were identified and reported in [21] that prohibits effective knowledge utilisation for decisions making during the integrated product design stage. Based on those shortcoming following requirements were identified to implement a framework supporting cooperative design modelling and manufacturing process planning:

- Usage of shared terminology to support communication between various stakeholders while supporting abstractions in model representation.
- Formalised knowledge representation to support verification and validation of system properties with simulation.
- Usage of temporal requirements specification for depicting time interdependency between certain inputs and outputs i.e. to depict operational mode requirements.
- Parametrised system model generation for producing various design configurations.
- Computational support for exploring trade-off space.
- Storing of knowledge learnt throughout the design process.
- Design parts reuse.
- Central design repository that can be accesses by various type of designers.
- Design decision logging and rationale traceability.
- Standard tool interface for various types of tools.
- Virtual collaboration of designers to support planning.

Some other issues related to computer based synthesis methods that are still present in the state of the art and are relevant to the scope of this research are [16]:

- Difficulty for practitioners to store their knowledge in required format.
- No automation present for guiding designers in following structured design methods without compromising their use of creativity.
- Scaling up of computer based synthesis system to support complex problem solving as per industrial scale.
- Integration of computational support systems with current design methodologies.

Some of these requirements are satisfied by present capabilities of tools and languages related to model based system engineering (given below) whereas some other requirements can be satisfied by current research in computational augmentation of model based system engineering

(MBSE) capabilities. This augmentation is being realised through the application of formal semantics on system model using semantically rigorous languages (such as OWL[3]- web ontology language) which enables certain degree of automatic reasoning. Following are the capabilities currently available in model based system development tools or are related to use of formal domain specific languages obtained by extending SysML semantics,

- Shared terminology is provided by system modelling language while enabling model consistency, completeness and well-formedness checking [22] [6] [9].
- Modelling language with formalised concepts and relations enable requirement verification through logical reasoning [14] or conversion of model developed in SysML into representation, such as Finite state process [23], that can be used for system verification for particular aspects like safety properties.
- Transformation of parametrised system model developed in SysML into analysis models for trade-off analysis while maintaining requirement traceability [24].
- Support for viewpoint generation according to various stakeholders needs by leveraging abstraction mechanism and semantic relation between inter and intra model elements [8].
- Support for model reuse especially for construction of structural, functional and behavioural models [25] [26].
- Documentation of design rationale as well as capability to reference certain analysis results for justification of design decisions in models developed by use of SysML [3].

Following issues that still present in current tools and languages or are not being researched in context of model based system engineering:

- They provide limited change propagation capabilities when design modifications are made due to discontinuation of part(s) from supplier (or because of unexpected result of verification activity or due to change in stakeholder requirement) [12] [15]. This limit is due to logical reasoning performed by the tools according to rules and constraints formalised in meta model, e.g. through the use of Integrity and completeness rules applied on model syntax [27] concerning system structure and stakeholder criteria (cost) but not concerning to functionality and behaviour of system with respect to physical laws applicable on system.
- Lack of application of automatic optimisation algorithm on SysML's parametric diagram representing mathematic model for dynamic analysis, which can require

[3] https://www.w3.org/OWL/

evaluation of change in input/output response based on change in component property values induced by the algorithm to satisfy constraints composed of weighted performance and other objectives as demonstrated in electromechanical design configuration optimisation in [28] [29].

- Inability to automatically utilise design rationale (or tacit knowledge) for selection of component's structural, functional and behaviour model such as use of fuzzy reasoning to encode design rationale in form of fuzzy rule [30] e.g. fuzzy rule related to stored models of dc motor- a dc motor is suitable for high output torque at low speed.
- System modelling languages doesn't provide support for representing temporal relations between model elements.
- Modelling tools also cannot perform automatic verification of functionality, and thus behaviour, modelled in models even at high level i.e. they cannot themselves explain (in qualitative manner) effect of change of parameter(s) on variable of interest which characterise overall system behaviour.
- Research efforts based on automating MBSE tool capabilities have not followed a specific MBSE methodology (which itself can be formalised using ontologies [19] for guiding developer) to implement automated capabilities and as a consequence they have not dealt with the interrelationships between those capabilities.
- The methods and techniques developed for computation design synthesis and configuration have also not been verified to work with heterogeneous systems of varying complexity (e.g. in terms of no of connections in the structural model etc.).

Following characteristics of a typical system model limits the number of the algorithms from AI field that can be applied to perform search, deduction and inference for planning actions to reach goal state (i.e. required design) while observing changes in environment or system model state (changes to requirements etc.) to update belief state or knowledge;

- System model is composed of highly interdependent model elements therefore it is difficult to perform operations such as requirement, structure and functional decomposition.
- Difficulty to apply reasoning (inference or deduction) because knowledge that is inferred or deduced may contradict or make inconsistent existing knowledge represented in formalised models.
- Search strategies requires use of negotiation techniques for resolving conflicts encountered in planning actions for producing structural and functional solutions for competing requirements.

- Multiple changes can occur downstream or upstream in the modelling process because multiple users interact with it to add, manipulate and derive/query knowledge simultaneously therefore in a domain with strong domain theory [31] all the change effects can be identified whereas for domain with weak domain theory all the effects are hard to predict which makes planning difficult.

Out of many research gaps identified above this research mainly focus on covering inability of current MBSE computational support methods and tools to confirm whether the model being developed is conforming to operational context, physical and technological constraints of the system application domain. Even though some level of checking w.r.t system domain has been made possible by use of DSML languages e.g. stopping distance of prospective system being modelled is achievable by given system configurations [32]. This same question can be further extended to know whether stopping distance of prospective system, based on currently modelled requirements, is achievable or not or if it is achievable then with what certainty.

1.3. Research Scope

This research deals with the system modelling activities performed during technical process phase (which is part of product lifecycle as specified by standard ISO/IEC 15288:2002 for System Life Cycle Processes shown in Figure 1-1) for creating models to support system specification, design, integration, validation (analysis and verification) and operation in the system lifecycle [33]. Technical process phase (as specified by ANSI/EIA 632 – Standard Processes for Engineering a System standard) is composed of stakeholder requirements definition, requirements analysis, architectural design, implementation, integration, verification, transition, validation, operation, maintenance, and disposal sub processes. This research only deals with modelling activities performed for requirement definition and analysis, architecture design and verification sub processes because these can be considered as a part of conceptual design stage where concepts are created, evaluated and selected according to the given requirements without physical implementation.

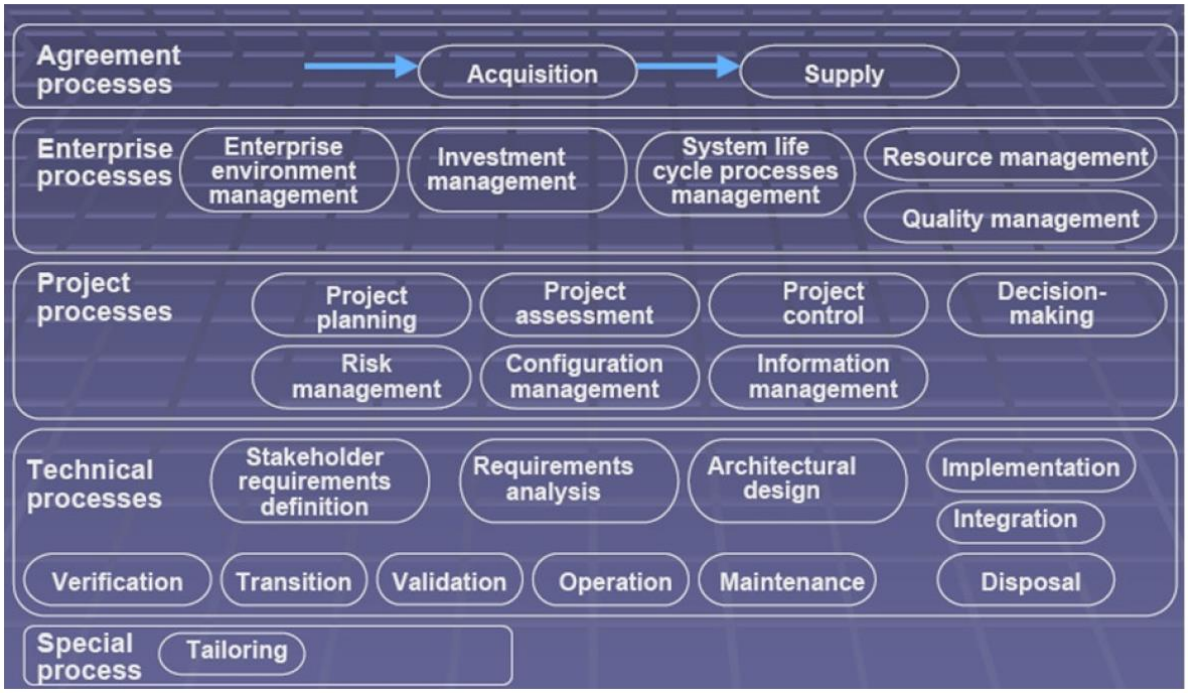


Figure 1-1 Framework of processes constituting product lifecycle as specified by ISO/IEC 15288 and ANSI/EIA 632 standards [33]

SysML provides suitable interface for capturing existing knowledge from system model developers as well as to describe new problem which can then be transformed into ontologies represented in OWL to enable automatic reasoning. Therefore this research assumes that it is possible to apply logical reasoning on formalised models to some extent on the bases of work from NASA/JPL team in formalising SysML models using semantic web technologies [22] and using techniques adopted from works on formalising requirements models through the use ontologies [34] and first order logic [35]. Similarly, transformation of some of SysML constructs into OWL (Web Ontology Language) is also achieved in [36] by defining explicit mapping between OWL semantics and SysML meta-model semantics whereas addition of domain specific concepts or modelling element and their semantics using profile extension mechanism of SysML is also a viable technique to convert SysML models into ontologies [37] [38].

It is also assumed that it is possible to express all types of knowledge (logical, numerical constraints etc.) related to system engineering concepts (e.g. relations between ports, function and component) as well as to domain specific concepts by using various logic formalisms (Description logic, first order logic etc.) and also possible to write rules for deriving new relations between the concepts in those formalisms based on works such as A-design [15] , design synthesis and configuration using constraint satisfaction [39], application of object oriented graph grammars for design synthesis [40] and other works using techniques like rule based expert systems, agent based methods, qualitative reasoning etc.

This research's main focus is to provide a framework which combines conceptual architecture generation and evaluation algorithm with a formalised model based system engineering methodology while leveraging existing general and domain specific knowledge obtained using formalised SysML models. Therefore, rather than focusing only on developing new algorithms for architecture generation which has been researched many times, albeit without consideration to scaling and industrial effectiveness as concluded in the survey on computer based design synthesis [16], a framework is proposed which can enable automatic reuse of model elements for model elaboration based on current model state as well as provide capabilities for functional and behaviour evaluation of model w.r.t requirements, available technology and physical constraints.

Even though proposed framework uses or suggest to use most of the technologies already been used in the area of computationally augmenting MBSE and design automation but utilisation of Deep learning technology in proposed framework is the novel one in computational augmentation of MBSE based model development.

1.4. Research Aims and Objectives

This work aims to formalise MBSE modelling tasks pertaining to Mechatronic system development for enabling physically feasible (w.r.t requirements, technological and physical constraints) descriptive physical architecture generation in terms of component types, component properties and component interconnections, by utilising the given requirements and existing solutions related to those requirements.

Following research questions can be formulated based on the given aim;

- Whether functional and operational requirements specified in a descriptive model (e.g. SysML model transformed into OWL model) can be used along with stored knowledge to produce physical system architecture?
- Whether it is possible to use data generated from simulation models (embodying technological constraints[4]) and simulation environment (embodying physical constraints) to infer actual components available to realise the generated physical architecture therefore scaling to complex systems?

To achieve aim and to answer the research questions following objectives are set:

[4] As new technology that can be deployed for use of human would have already undergone virtual as well as physical V&V as well as existing systems could also be utilising that technology but may be in different application domain

1) Investigate method of organising and reusing knowledge related to system's structure, function and behaviour for performing automatic descriptive model elaboration through model element inference as well as element to element relation inference.
2) Investigate use of AI technologies to determine whether or not system with specified input to output response under given operational requirements can be realized under known technological and physical constraints.
3) Investigate methods to discriminate between different design configurations with different specialisations (e.g. configuration with least power consumption or high power output) based on imposed design constraints over system properties.

1.5. Chapters Overview

Chapter 2 presents review of methods and tools used in supporting system engineers with design knowledge utilisation through formalisation of system modelling language, common modelling patterns utilisation for model retrieval, requirements formalisation techniques, conceptual design synthesis, domain specific simulation models retrieval, system model development tools interface with domain specific tools etc. This chapter also briefly reviews basics concepts of machine learning, auto-encoder neural networks and variational auto-encoders.

Chapter 3 presents problem formulation of outputs of requirement definition, architecture design and analysis stages based on MBSE methodologies review and then introduces the framework with introduction of its main components and finally finishes with introduction of system application domain and domain hierarchy concept.

Meta-Models required for constructing descriptive system model and retrieving existing instantiated descriptive models are presented in Chapter 4 along with some exemplary rules and constraints required to automatically generate a physically feasible Mechatronic system. This chapter then ends with demonstration of use of domain specific knowledge in form of 2-D semi-qualitative kinematic framework for generating rigid kinematic link chains.

Retrieval of simulation models, representing retrieved descriptive designs in different scenarios, based on given I/O requirement behaviour is presented in Chapter 5. This chapter demonstrate novel application of neural networks in retrieving designs based on their topological and input conditions similarity.

Chapter 6 compares research objectives against the work presented in previous chapters while highlighting shortcoming of proposed framework as well as of experiments conducted in Chapter 4 and 5 to give direction for future work.

Chapter 2 Literature review

In section 2.1. overview of general MBSE methods and its characteristics is given followed by discussion on techniques for producing computer understandable system engineering models in section 2.2. In section 2.2.2. , general approaches in the field of design automation has been reviewed independent from their application in computationally supporting MBSE. Last section then introduces basic machine learning concepts while differentiating between supervising and unsupervised learning and then finally VAE (Variational Auto-encoder) is briefly reviewed.

2.1. System Engineering and MBSE

Increasing complexity of system models requires consideration of inter and intra relationships between requirements, physical components, environment, users and between their various properties according to constraints imposed by requirements and physical laws. Because of consideration of wide ranging aspects in models, development of complex system models is now being supported by system development methodologies to ensure efficient management of information related to the system and development process.

A brief description of terms such as methodology, development lifecycle and acquisition lifecycle is given below. Defining characteristics of MBSE methodologies are then highlightened using an information model. Introduction to different MBSE methodologies is provided in section 8.4. of appendix. This section is adopted from survey on MBSE methodologies [33].

A methodology tackles a particular class of problems that entails common characteristics using collection of methods, tools and related process [11] [33]. A method is a set of related activities, techniques, and conventions that implement one or more processes and is generally supported by a set of tools. A model-based systems engineering method is a method that implements all or part of the systems engineering process, and produces a system model as one of its primary artefacts [41]. Whereas an *Environment* encapsulate, integrate and facilitates the use of methods and tools.

System Engineering (SE) development process models are part of the acquisition lifecycle models and describes the activities required to be performed to implement system engineering for system acquisition. Lifecycle development process models i.e. Royce's waterfall model, Boehm's Spiral Model, Forsberg and Moog's Vee Model are used for incremental and iterative development in software development whereas Vee model's variations have been applied to general system development involving mechanical, electrical and software systems. MBSE methodologies are also built upon these lifecycle models. Difference between acquisition and development lifecycle stems from the definition of the acquisition i.e. multiple application of

development lifecycle is a key part of acquisition apart from its other constituents i.e. concept generation, required technology development and system manufacturing and deployment.

ANSI/EIA 632 standard formalises and defines top level practices for development lifecycle and requires other low level standards to elaborate those practices. It also provide set of processes required for system development. IEEE 1220 standard implements or elaborate those practices defined by EIA 632. IEC/ISO 15288, shown in Figure 2-1, define both general and domain specific system lifecycle processes i.e. by providing a common framework for describing system lifecycle.

Figure 2-1 ISO 15288 system lifecycle [33]

Difference between MBSE and traditional SE standards is that the MBSE methodologies do not follow pure waterfall model and activities of MBSE methodologies are completed in terms of adding detail or by developing detailed models incrementally [33]. In MBSE, the sub processes (development of required technology & alternatives identification, models building, existing analyses data gathering, models validation against data and verification against requirements) in different project phases (system definition, preliminary design, detailed design and design qualification) can be repeated many times.

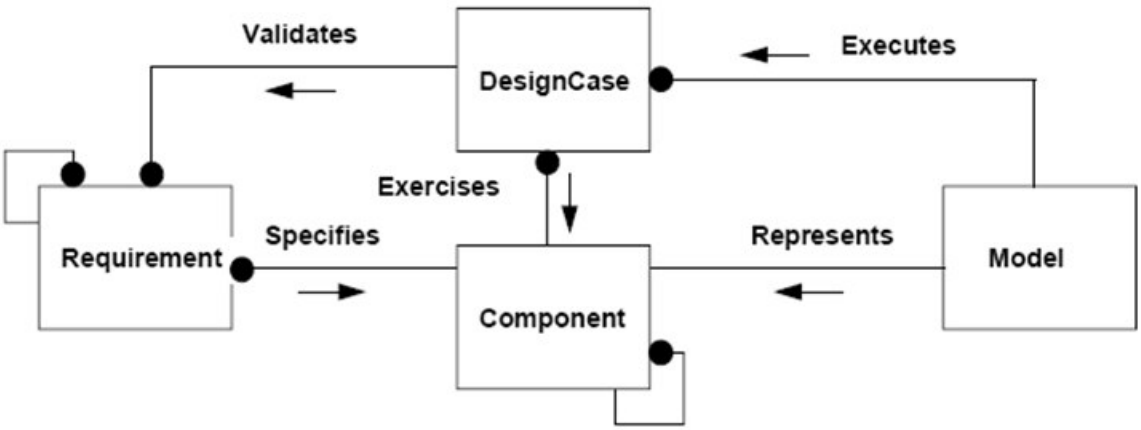


Figure 2-2 Information Model for Model Driven System Design [42]

An information model for Model Driven System Design (MDSD) is represented in Figure 2-2 which specify relationships between concepts found in MBSE context e.g. a model represents component which can be decomposed into other components, model is used to execute (simulate) design alternatives through parameterisation etc. By examining the information

model it can be said that model based system development process is not sequential but iterative and require execution of parallel activities because models are required to be developed for calculating requirement specifications for components through trade-off analysis over design cases (i.e. parameterised models) by using the abstract (user) requirements given at the start. As Baker et al. [42] states that, "in early states, the models are low fidelity and geared towards decision making; eventually, models become sufficiently faithful for compliance assessment" which, along with information model, highlights the incremental nature of the model based system development and advocates for performing various parts of the process in concurrent fashion as they all inform each other.

Blackburn et al. [5] conducted survey to assess technical feasibility of using models in a holistic way to support product lifecycle and provided recommendations in support of partial system integration and testing throughout a continuous (incremental) integration approach which tightly couples two sides of "V" process model using semantically formalized interfaces in behavioural and structural terms i.e. for descriptive and computational model integration.

Besides being iterative and incremental, system development process is also dynamic and distributed. It is dynamic because multiple changes occurs at different points of the process (could either be at requirements side or verification side or modelling caveat found during detailed design) and it is distributed because multiple users interact with the central design repository to add, manipulate and derive/query knowledge simultaneously.

In this section MBSE methodologies will be reviewed briefly and one will be selected. The tasks that selected methodology prescribes are used in Chapter 3 to formulate problem definition for system model development.

2.1.1. Brief MBSE Methodologies Review

It can be deduced from all the methodologies reviewed in section 8.4. of appendix, except Vitech MBSE methodology, that specification of system is based on high level functional requirements given in requirement model. Therefore the physical architecture achieves this high level system functions while satisfying safety, performance and other requirements. The same can also be seen in research efforts that aims to automatically deduce system structures [30] and configurations [15] to achieve high level functions. Therefore most of the system development methodologies follows approach of iterative refinement from conceptual level to realisation level.

The Vitech's methodology is distinct from all other methodologies as it follows a parallel process model unlike others that uses "V" process model. This methodology is most suitable for established system as it allows system engineer to relate available physical architectures

with given requirements at high level and then apply modifications as required in subsequent iterations.

Harmony SE and Object Oriented SE (OOSE) methodology follows the traditional "V" process model and it can be said that Harmony SE is a more refined version of the Object Oriented SE methodology with dedicated tool support. The prime difference is in the top down and bottom up approach prescribed for design phase and in integration phase of Harmony SE respectively, rather than following only the top down approach as prescribed by OOSE. Harmony SE also entails iterative cycles during both phases therefore supports incremental model refinement from high level models. One advantage Object Oriented SE methodology has that it prescribes requirement variation analysis during requirements validation to estimate requirements variation due to potential changes but it does not specify a particular method for doing it.

Vitech as well as Arcadia gives the flexibility to use any type of diagrams in various phases whereas the Harmony SE and Object oriented SE methodology requires usage of diagrams in a particular sequence according to the development phase. One sequence given in Harmony for System Engineering Deskbook [43] is to create use case scenarios first in sequence diagrams using use cases from requirement phase and then derive participating functional flows, after which sequence diagrams can be used to identify types of ports and interfaces to be used in internal block diagrams between actors and use cases. Finally state diagram can be used depict state based behaviour.

Vitech and Arcadia both leverage the benefits of using semantically formalised DSMLs (Domain Specific Modelling Language) which are specialised from SysML. Tool associated to Vitech's methodology extends SysML syntax and applies formal semantics to it for enabling consistency and completeness checking of models elements and of relations between elements of requirement, V&V, Physical architecture and Behaviour/functional models. It also provide methods for various types of system model testing. Tool associated to Arcadia methodology also extends SysML to create its own Domain Specific Language to provide well-formedness, consistency, completeness and coverage checking with respect to the DSML's meta-model.

Drawback of Arcadia is that it does not clearly defines boundaries between the development phases as seen in the overlapping of the system need analysis phase with the logical architecture development phase as both phases focuses on describing interactions between functions and actors as well as on performing functional decomposition to allow function to component allocation.

2.1.2. Key MBSE Tasks Amenable to Intelligent Computational Support

Models enables incremental detailing of system requirements and architecture by using abstraction mechanism e.g. refinement of requirements from the user level to system level, specifications of interactions obtained from use case into interface specifications etc. In general, the objective of system model development is synthesis of balanced system design that satisfies performance and operational requirement while meeting design constraints such as size, power consumption, cost etc. [3]. Main output artefact of system design synthesis is design specifications (i.e. component and interface specifications [43] e.g. hardware component dimensions, its interface properties, I/O parameter characteristics etc.). The process of generating and verifying these specification includes detailed design analysis which in turn generate other artefacts e.g. mechanical CAD of system's component and connections. The synchronisation between these artefacts (test cases obtained from use cases and requirement and behaviour analysis results) is also maintained through the synchronised updates of the models. The developed models and generated artefacts are stored in a library for reuse.

As we are mainly interest in automating model development process in order to generate physically valid design specifications or at least physical architecture therefore key tasks required to algorithmically (partially) implement a MBSE methodology (especially Harmony-S.E methodology) are given below. Harmony-SE methodology is described in detail in section 8.4. of appendix along with SysML diagrams. These tasks assumes we are dealing with well-established[5] mechatronic system and goal is to obtain physical system architecture and design specifications starting from user requirements:

1. Automatic classification and decomposition of User requirement to System requirements requires:
   i. Formalisation of requirement types and relations among them.
   ii. Building use cases based on external environment/user interactions.
   iii. As-is system retrieval from similar requirements.
   iv. Knowledge of up to what point decomposition needs to be performed which require use of something similar to functional basis (see section 2.3.1.).
2. Generation of logical and physical system architecture:
   i. Further decomposition of functions up to physically realisable operations.
   ii. Inference of functional flows using use case requirements and using decomposed functions and their operations.

[5] Similar system has been developed before and we are not dealing with creating entirely new system.

iii. Identification and allocation of high-level interaction interfaces, connections between them and other properties to existing physical subsystems.
iv. Derivation of behavioural physical model using input/output parameters captured in functional flows and subsystem properties and interconnections.

3. Allocation of physical subsystems to known physical components:
   i. Grouping or decomposition of logical/physical subsystems so as to match their functionality with functionality delivered by available components.
   ii. Selection of physical components for function realisation and deduction of correct topological arrangement for them.
   iii. Elaboration of subsystem interfaces and therefore interactions between the components using component specific operations/methods and properties.

The problem formulation formalising outputs of system modelling tasks in Chapter 3 mainly follows above sequence of tasks.

2.2. Computational Augmentation of MBSE

Authors in [5] have surveyed use of MBSE in industries and stated that one of the emerging idea in applying model based support is "Computer augmentation, where digital assistance will begin to understand what we are trying to model and through advances such as machine learning and integrated visualization can act as a knowledge librarian helping us to model some aspects of the problem or solution at an accelerating pace"

Efforts to utilise design knowledge in form of models and verifying consistency between the reused knowledge (or models) and models created for design problem in hand requires assistance of tools to perform various checks. Current model based system engineering (MBSE) modelling tools affords capabilities such as ensuring consistency between information encoded in the models, requirement tracing through use of formalised relations, reuse of partial models stored in libraries, enforcing of methodology constraints (sequence of model creation, type of content required in different model types etc.) [43] and validation of model construction with respect to modelling language's syntax formalised in meta models and also with respect to its extension with domain specific concepts and relations between them e.g. a 'connectedTo' relation can only connect power source and power distributor concepts with multiplicity constraint of exactly one on both concepts. Therefore formalisation of descriptive models has been mainly done based on applying semantics using domain specific concepts and relations to system modelling language to ensure consistency and integrity of models with respect to domain constraints.

MBSE tools, such as Vitech's Core[6] System modelling environment, uses semantic relations between the cross diagram model elements (such as "isverified" relation between a requirement and a physical architecture component). Such tools also provide model checking rules for checking consistency and completeness in which they check for relation's domain type, range type, quantity or multiplicity constraint and other constraints related to its type (a functional relations can only have one range individual related to one domain individual) defined in meta model. A model is complete e.g. if all its relations has correct domain and range individuals, whereas it will be inconsistent e.g. if any constraint related to an element's definition is violated i.e. quantity constraints.

Another system modelling tool, called Capella® [27], uses different types of rules such as integrity or conflict checking rules - a component has to have attributes of name or id and cost type, well formedness rule – state diagram can only have one initial state, coverage rule – an interface has to be used by at least one component, completeness rule – Interface has to be implemented by at least by one component.

MBSE is used in large scale projects with custom made tools, such as Virtual Spacecraft Environment for Astrium Satellites [11], Melody Advance from Thales Alenia space [11] and Model-based Engineering Environment (MBEE) from Nasa/Jet propulsion laboratory [5], which uses domain specific semantically formalised concepts and relations. Similarly, work from Integrated Model Centric Engineering [11] has developed ontologies, based on typically used modelling patterns, through semantically formalising subset of SysML constructs to enable generation of trade space for analysis activities.

DEVICE framework [44] [11], being developed at Thales, aims to support design, reviews, integration, testing planning and evaluation by providing modelling tools that follows MBSE methodology called ARCADIA. It aims to develop web technologies to support consistent information exchange, coherency and consistency among descriptive and simulation (computational) models. Partial development of DEVICE framework is presented in [11], where framework is developed to exchange data through the use of web technologies between domain specific analysis models and models developed in system modelling environment for purpose of trade-off studies and system verification in all modelled scenarios. The framework integrates system model with optimisation, sensitivity and reliability analysis tools for analysis of effects of uncertain qualitative and quantitative factors over deterministic simulation models response. The framework is formally defined by defining concepts and relations between elements of domain specific simulation models related to different analysis software and

[6] http://www.vitechcorp.com/products/core.shtml

between elements of modelling language used in a system modelling environment using conceptual data architecture. It also supports automatic pre and post data processing for adopting data to formats required by various simulation software and modelling environment.

It is also attempting to incorporate interactive augmented reality technologies with physics simulation for help in validation of various types of requirements e.g. operational, integration, testing etc. However the tool being developed only provides integration between other tools and does not have deliberative reasoning capabilities on information being exchanged.

2.2.1. Use of Semantic Web Technologies for MBSE Support

Emergence of Semantic Web [45] and its associated tools like (OWL –Ontology Web Language for knowledge representation, SWRL – Semantic Web Rule Language for rule definition used in reasoning and SPARQL – Knowledge base query language) provide features such as consistent knowledge sharing and automatic knowledge discovery [20] which has inspired researcher to capitalise domain knowledge in form of knowledge base [10] & [18] to support system engineering activities like requirement definition through the use of ontologies [46] or using formal logic language (i.e. First order logic) [35].

Graves [36] has used semantic mapping approach to leverage formal semantic rigor of OWL2 and graphical interface of SysML by translating subset of SysML's constructs (block definition diagram) to the former and then extending the transformation to cover more constructs using a logical system called type theory as given in [47]. According to author, a system and environment model developed for particular type of simulation (or analysis) using formalised (by type theory) SysML can be queried for answers to questions such as whether or not an airplane (described in the model) verifies the capability of taking off and landing for given weather condition. Similarly, [48] has used RDF graphs to represent moderate number of requirements and component attributes and interfaces in a library for use in automatic generation of a feasible system configuration by connecting compatible interfaces and to perform trade off analysis for component's attribute selection in an evolving design environment.

Team at NASA/JPL [22] has created ontologies for subset of system engineering concepts like view, component, function etc., and to develop formalised vocabulary and rules for checking well-formedness (e.g. presence of correct relations among instantiated components) and consistency (in usage of interconnected model elements in different views) through automated reasoning on their formal representations. These ontologies are mapped to SysML modelling entities (i.e. blocks and dependency relationships) by loading them to sesame repository and using SPARQL queries to create digests which are then transformed to SysML profiles (used

to extend UML or SysML Meta-model semantics for domain specific needs) using Query/View/Transform (QVTo)- Operational language.

Same team has also leveraged reasoning capabilities of ontology by defining semantics of state analysis methodology (used for control system development and documentation) using ontologies and projecting those semantics on custom made SysML stereotypes for checking conformance or compliance of architectural and behavioural models (developed using the stereotyped SysML elements) against the semantics of state analysis methodology [37]. However they are not able to transform all the constructs (used in construction of main 9 diagrams for requirement definition, system specification and behaviour/functional verification).

Same language extension technique is also used in [38] to verify interface compatibility and conformance of structural model against functional requirement specification during a subsystem or module replacement. Profile called SysML4*Mechatronics* is used for extending semantics of some of SysML elements to define domain specific concepts [38]. This profile extends the SysML concept of port and flows to include domain specific ports i.e. mechanical port, software port etc. Model developed using the stereotypes, defined in profile, is then transformed manually into an ontology which is queried using custom rules (formed using SPARQL) to ensure compatibility between the port types and its attribute directions. The SPARQL queries are used to verify interface compatibility by checking matching between the ports and port attributes of the replacement module with the module being replaced and with modules to which it will connect. It also checks impact on the functionality of modified system by matching the provided and required operation attributes (related to software port) of existing module with operation attributes of replacement module's software port as well as by matching the functionality attribute of the modules being exchanged. This approach doesn't assess overall system's behaviour (i.e. the input and output behaviour) integrity when a subsystem is replaced and its future work recommend to look into component specific attributes for checking operation-ability of system by determining component specific requirements (like power consumption requirement of replacement). This will also require finding impact of replacement's requirements on overall system's operational requirements (e.g. change in system's power consumption in various operation modes).

The ontologies developed by NASA and TopQuadrant® for NASA's Constellation Program specifies the types of quantities, units and value and relations between them. This enables automated checking of correct property value unit and value type usage as well as transformation to an XML format (using OWL compliant XML schemas) to enable interoperability between modelling tools [49].

The formal knowledge can also be transformed into domain specific representations, which unlike SysML are easier for domain experts to understand. Domain specific languages based on process and product meta-model (similar to one described in [14]) can enable designers to easily construct different types of models e.g. a model showing component connections and their attributes and model showing process states as demonstrated in [14]. The domain specific models can then be transformed into parametrised analysis models (similar to mapping of ontologies to state analysis tool in [50]) as well as into formal knowledge representations (e.g. first order logic in case of [14]) for consistency and completeness reasoning, similar to the reasoning performed by current MBSE tools as described above.

2.2.2. Patterns in System Engineering

Concept of patterns in software engineering is very common because they provide skeleton code which can be specialised according to specific aspect of the reoccurring problem.

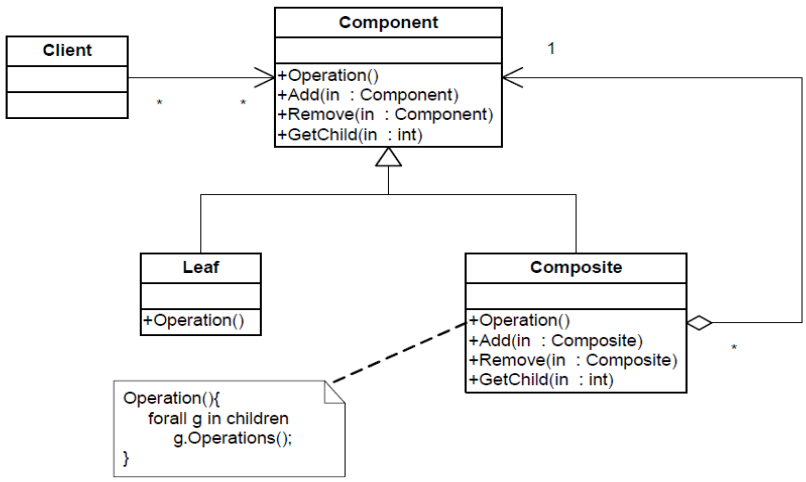


Figure 2-3 Composite pattern represented in UML diagram [51]

Object oriented programming uses composite pattern (as shown in Figure 2-3) by renaming the classes with respect to problem domain entities which have similar type of relations between them. The concept of composition is also very familiar in system engineering, however the operations implemented in the software composite pattern are only implemented in terms of constituent members but not using their physical properties which would be the case with the electromechanical assemblies.

A design pattern captures designer's knowledge or design rationale [51], which helps to better predict the course of actions and their outcome by reflecting upon previous mistakes. Patterns captures the design rationale behind evaluation and selection of solutions related to a recurring problem [52].

Approach taken in [26] also shows how productivity can be increased by using design patterns to support model reuse. Design patterns used in the approach captures the problem context description along with the intended functionality of models it refers to and model's further decomposition. Design patterns also contains reference to solution patterns which contains structural solutions embodying the intended function and reference to partial simulation models. The solution patterns also store rationale for their usage (in terms of advantage and disadvantages of components in particular situation). The structural model for solution is stored as IBD which is transformed manually into a partial simulation model by using the flow properties related to the input/output port (i.e. input/output variable types). This partial simulation can then further elaborated manually based on specific requirements. Such design patterns links functionality, behaviour and structure with the usage context using design rationale which is suitable for retrieving models from a knowledge base for particular problem.

Retrieval of model require matching (either fully or approximately) their structural, behavioural and semantic aspects with given incomplete model [51]. Retrieval of design pattern can be performed based on matching structural aspects such as relation between classes (i.e. generalisation, association etc.), attributes and methods. Semantic matching can be used to match structural aspects and involves checking for type of relations between objects, checking composition of name to identify the role that a particular class plays etc. Matching of behaviour aspect is normally used to increase confidence in the similarity of the retrieved artefact therefore uses result of structural aspect matching. The behaviour is determined by method invocations representing interactions through message exchange which could be different in different scenarios therefore required matching can only be made by analysing runtime behaviour if invocations cannot be used to estimate the behaviour. Dynamic matching entails checking for features such as sequence of class's method execution or interactions, effects of interactions/execution on other objects etc. This type of checking can be performed using sequence diagram describing interactions.

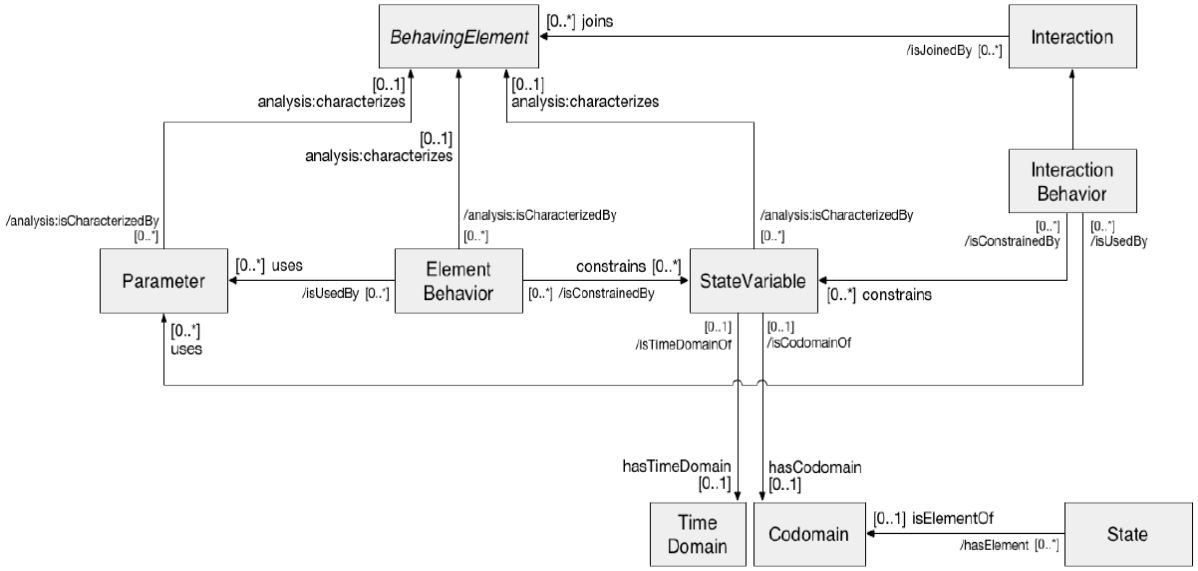


Figure 2-4 Behaviour pattern defined in Behaviour ontology [50]

Design patterns (as shown in Figure 2-4) can be used as a template for a problem specific model construction and can be embedded as stereotyped elements in SysML using the profile extension mechanism of the language [50]. Any model element used in a new model has to be an instance of such a template. Use of patterns, defined in ontology, can be used to organise and retrieve models during model reuse. A pattern used as a template for creation of several stored models becomes an index which can link an incomplete model with several solutions or several completion branches. This approach is also used in [53] [54] to organise and retrieve abstract simulation models instantiated from a template, also called the Structure-Function-Behaviour-Interface-Operation (SBFIO) domain model. Further references related to pattern based model retrieval techniques can be found in section 8.5. of appendix.

2.2.3. Requirement Formalisation

[55] has developed ontologies using first order logic representation which provides semantics to a common terminology related to engineering design process. The semantics are applied using sets of axioms which enable deduction of further knowledge, conflict and integrity checking (e.g. correct type is used) by defining and constraining the terms in the terminology. Requirement relations such as decomposition, derived and explicit along with classes of requirements, such as structural, performance, functional etc., are formalised. The ontologies are defined and validated based on questions they are intending to model. If question such as, i.e. whether satisfaction of child requirements leads to the satisfaction of parent requirement?, is defined well using axioms (constraints defined on concepts or terminology in ontology) then logical reasoning on that axiom should produce the expected answer. Relations between requirement, part, features and constraints has also been formalised to enable requirement

traceability, consistency and satisfiability checking as well as allowing requirement change management.

Authors in [35] has created a requirement meta-model which contains commonly used requirement relation types formalised by defining and constraining relation types (contain, refine, conflict and require) as intensional (concept defined using other concepts) and extensional terms (i.e. concept defined using primitive facts or types) in first order logic formula relations and in rules of set theory. These formulas and rules formalises properties of relations (reflexive, symmetric and transitive) which enables consistency checking of the requirement model and inference of new relation types using the initial requirements as well as finding effect of change in one requirement on other. The requirement relations are used as input to a logical reasoner as two different types of facts, representing requirement relations (or relation between required system properties) as first order logical sentence and as relation among sets e.g. the R1 refine R2 relation for the system S1 and S2 satisfying set of properties P1 and P2 of requirement R1 and R2 is given in set representation as S1 subset of S2 and as logical sentence *(P*1 *all-in-whole P*2*)* ∧ *(P*1 *some-implies-in P*2*)* which means some or all properties in P1 should be same as all properties in P2. Reasoning is applied over the input i.e. requirement document , by encoding it as requirement model which is an instance of the meta-model created by mapping first order and set theory representations to OWL. To use this approach, requirement engineer is require to learn first order logic to formulate requirement model conforming to the meta-model proposed in the approach.

Another method of formally representing requirements is given in [56] by recognising the system, about which requirement is being made, as black box which provides relationship (which can be characterised by quantitative or qualitative attributes) between its inputs and outputs. The requirements format corresponds to this 'black box' view of system where input and outputs relate to external entities effecting or effected by the system whereas the relationship represents high level system function (operate, interface etc.) which is characterised by quantitative or qualitative constraints over the relationship's attributes. Further specification of such requirements means decomposing high level subject system into logical subsystems which in turn will have more specified requirements. However, as Author has indicated that design constraints cannot be represented in this formalism neither the requirements about characteristics of external interfaces and external interactions. Author has suggested to use other type of models to represent such requirements e.g. stakeholders and needs model, functional interaction model, state model, feature model, domain model and logical architectural model. For this purpose author has proposed S* meta-model.

The pattern based system engineering methodology (PBSE) [57] uses S* meta-model (shown in high level in Figure 2-5) to impose model construction rules on construction of various types of requirement and operational scenario models using formalised concept and relations defined in meta-model. This meta-model, like others, can also be specialised for specific domains which can then be used for modelling products. The models constructed for capturing product family requirements can be parametrised in terms of functional roles and feature attributes to produce variations upon the base requirements model so as to capture specific requirements related to particular products in the product family.

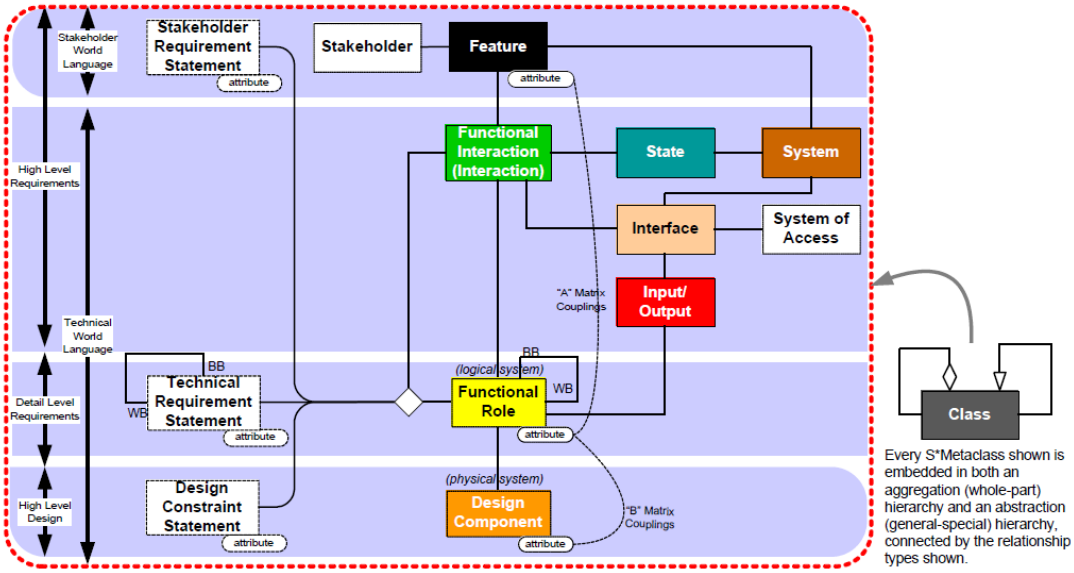


Figure 2-5 S Metamodel summary [57]*

Similar approach is also taken in [9], where Semantic Product Modeling Language (SPML) is developed for use in product modelling to enable capabilities such as test case generation and requirement traceability. This language usage is similar to SysML but has more semantical rigor and also provide domain specific concepts and relations. Although it links the requirement, function and structure using concepts such as form and feature it lacks constructs to represent function sequencing and behavioural interactions which can be represented in sequence, state and activity diagram in SysML.

2.3. Conceptual Physical Architecture Generation

Design specification, synthesis and configuration is part of System architecture design and design synthesis phases of every MBSE methodology. These phases are preceded by requirement definition and analysis where user requirements are translated into system requirements. During these three phases, a system engineer corroborates information provided by experts from different fields to produce physical architecture that meet the operational and performance requirements. This physical architecture is mainstay of the trade-off studies which

are performed next in order to select the balanced configuration that satisfies design constraints while meeting performance targets as closely as possible. From here on conceptual design refers to physical architecture.

Automated design can be divided into two parts based on knowledge required according to [39], namely automated design synthesis and design configuration. The latter activity is performed where existing knowledge, such as range of property values of components, connection compatibility between components, etc., is available whereas the former activity deals with situations where existing solution to the design problem is not available, but only generic concepts, their relation and constraints on those relations are given. Works in design automation has aimed to automate these two activities by using different type of knowledge representations such as graph grammars, first order logic, genotype, ontologies, bond graphs, model fragments etc., therefore mostly relying on use for rule based techniques inference of relations and in some cases neural networks (e.g. in case of NETSYN) were also used.

Conceptual design requires consideration of factors from different domains affecting system's functional behaviour in different operational scenarios. System engineer(s) are responsible for designing conceptual and physical architecture of the system but considering all the details regarding the characteristics (physical and functional) of domain specific components and their interfaces in system model is very difficult just by system engineer without input from experienced mechanical, electronic and software developers or from knowledge repositories storing such experiential knowledge.

[16] "Computers can help designers explore new directions by providing a wider variety of possibilities during routine design. In this case, computers can automate tasks, leaving more time for creative activities, and help reduce errors, thus improving value".

Decisions taken during system conceptualisation stage can also have substantial effects down the system development process, especially in terms of cost [39, pp. 1-2]. Therefore, evaluation of conceptual architecture, in terms of functionality, operational range and structure that can realise this architecture, is also very important.

The approaches reviewed for conceptual architectural generation uses techniques which takes abstract representation of problem and knowledge so as to generate abstract problem specification alternatives and solution alternatives to those alternative problems.

2.3.1. Design Synthesis

Survey over the computer based design synthesis methods [16] has reviewed function based synthesis methods and analogy based synthesis methods. The function based methods provides

task knowledge, such as decomposition knowledge, which helps the designer to decompose required high level functions into standardised interconnected decomposed functions through the use standardised flows that can be realised by hardware or software components. Weakness of this methods highlighted by authors is that the high level functions that are given as input to the designing system may not capture all the subtleties of the requirements which is obvious from a system engineering methodology perspective which advocates for use of different types of requirements. This weakness often leads to generation of undesired solutions and which then expands the design alternative space making it difficult to find optimal solution.

The other approach reviewed focuses on use of in-domain (also called case based design) and cross-domain knowledge elements such as design problem, solution, domain and strategy (or mapping) that can be used to obtain solution from the problem. The relations between such elements are often formulated using abstract conceptual schemas (or meta-models) which facilitates cross domain knowledge transfer i.e. from source which provides abstract solution for the abstract problem identified in the target domain. Weakness of this method is related to difficulty in automatically retrieving effective analogical solutions and to automatically adapting those solutions according to the problem because most of the methods developed for analogical or case based design synthesis requires designers to adapt the solutions manually due to specific context dependent variations in problem.

An automatic assembly design system is described in [30] which performs conceptual mechanical design synthesis and preliminary design alternative generation by instantiating mechanical model from a conceptual schema and elaborating it by using analogical reasoning applied on case based knowledge storing functional, structural and behavioural models. It also checks structural assembly of the model using qualitative reasoning applied over bond graph representation, whereas selection of physical components is performed using fuzzy reasoning to replace the generic components in the model. The mechanical model is composed of other models such as function model made up of functional carriers (to allow function allocation to components), structural model describing component interconnections and behaviour model depicted using bond graph. Feature links are used to enable correspondence between components in structural model and their geometric counterparts residing at different level of abstractions.

The system is divided into three parts to aid the designer i.e. supervisory controller called Design planner which generates the domain-independent design model by creating logical relations between ‘functional carriers’ i.e. components representing required functionality, Design consultant which provides knowledge associated to functional carriers e.g. similarities to available structural solutions, analogical design rules representing input/output patterns

extracted from mechanisms with similar input-output relationships which helps to consider external input effects (such as forces) on structure of new design, related behaviour models etc., and Design sketcher that provides geometric drawing capabilities. This system uses a shared design repository which stores design model (i.e. hierarchical design object) in form of 'multi-graph data structure'. Case based indexing and similarity matching, is based on the function and relationship between function, behaviour and structure models of stored designs respectively. Fuzzy values are used for incorporating fuzzy requirement criteria and importance weighting for selection of specific components (in terms of their attributes) in place of generic components used in the structural model.

Although some of the capabilities of this system are nowadays provided by system modelling tools, such as use of unified representation consolidating various design aspects, but other capabilities related to use of knowledge base system are still absent e.g. design process recording in the multi-graph data structure where each data node represents design at different level of detail therefore recording the design path in node relationships, alternative design generation using functional substitution or allocation (in system engineering terms) and feature inheritance representing high level geometrical constraints inherited by low level components when structurally elaborated with additional features. The feature links are used to maintain consistency between the initial conceptual structure represented using abstract shapes (e.g. shaft represented as line) and more detailed structure (e.g. shaft represented as 3-d cylinder the length constraint stays same).

Implementation of all the parts of the proposed approach has not been performed however proposed approach is consistent with new tested approaches such as use of fuzzy reasoning for component selection based on requirement criteria and requirement importance in hybrid decision support approach [13] and use of qualitative reasoning over bond graphs [58] for generating functional configurations of electromechanical systems.

Similar approach is used by [40] to construct abstract physical architecture where high level functions are decomposed into sub functions by using their input-output ports and are represented as reconciled functional basis i.e. generic functions corresponding to bond graph elements such as convert, supply etc., and network of physical effects (physical laws represented as blocks with ports) is allocated to those sub-functions and structural components by matching their input/output port types (such as rotational, translational etc.). The complete physical product architecture is built using design rules which relates function, structure and physical effect elements using formalised relations. The Object Oriented Graph Grammar (OOGG) knowledge representation (a type of rule based production systems [16]), used in the approach, is composed of the vocabulary and graph transformation rules. The vocabulary is

defined using meta-model that defines hierarchies of node types in terms of port-based generic function, behaviour and structural elements, edge types (in terms of relation e.g. decomposition, realisation, flow type etc.) and port types (in terms of flow and domain type e.g. rotational, electrical etc.). The OOGG uses graph transformation rules (can also be called problem solving or task knowledge) to guide automated conceptual design process by defining admissible connectivity between the meta-model elements. The meta-model is also specialised by incorporating vocabulary or terminology related to concepts of particular application domain.

Four generic type of rules are given for high level function decomposition into sub-functions and behaviours, creating sub-function and behaviour chains to match high level function's input and output port types, physical component assignment and merging of redundant components. These rules first determines the validity of rule application by matching their antecedents (stated in terms of generic node and edge types) with given graph pattern and then search for sub-functions and behaviours with similar port types for asserting relations (function realise effect, function decompose to sub-function etc.,) between them before returning the graph for application of next rule. Application of rules is controlled by providing Boolean and integer parameters constraining or acting as additional knowledge of desired solution to guide the search (to constrained random walk search) which adds and deletes edges and nodes according to match between rule's consequent and the given graph.

The assignment of behaviour or physical effect to decomposed functions or functional basis is realised by matching abstraction ports on the physical effects and on the functional basis. Abstraction ports are manually assigned to decomposed functional basis given in the meta-model and are based on the energy domains of physical input and output ports i.e. a gyration port is assigned to "convert" function as it converts rotational mechanical energy to electrical energy. The assignment of abstraction ports to physical effects is undertaken by identifying input and output variable types and their causal orientation i.e. direction of effort and flow variables, from the mathematical equation associated to the physical effect. This method is not generic because special cases has been considered due to the use of variable direction instead of using the mathematical relation type i.e. proportional, derivative etc., to assign the abstraction port types between the variables and their domains. Based on the assigned abstraction ports, bond graph elements representing those ports are then retrieved to build behaviour model.

Design evaluation is based on the definition of correct relations between port types in the meta-model without consideration of correct input to output causal relation of the overall functional chain i.e. without checking its operational feasibility. Instead of formalising SysML relations author has introduced non-standard relations or edges in the used meta-model therefore

genericity is not guaranteed. Similar issues are also highlighted in survey of computer based design synthesis methods which states that there is no formal language to describe graph grammars and the ones that are used cannot sufficiently represent level of knowledge complexity as required in engineering design [16].

Another latest effort [39] applies Binary satisfaction algorithms to generate valid topologies in conceptual design stage for given input-output requirements. It also uses constraint satisfaction technique to generate valid sets of parameter value assignments and then uses simulated annealing to find an optimal set of parameter value assignment to minimize an objective function. The automatic generation of alternative concept topologies (configurations) is based on meta-models composed of port, element and signal type hierarchies.

A meta-model consisting structural components, ports and boundary elements (encapsulating the system of interest while providing the input/output ports related to the I/O variables given in requirements) is used as generic knowledge. It also contains port type hierarchy (containing signal and energy ports) along with their associated attributes such as port direction, permitted range of effort and flow variables etc. The structural elements in the meta-model are used for automatic generation of partial simulation (bond graph) models which are terminated with open connections and therefore can be used in a model based approach for integration with other partial models. The ports of the components embodies the bond graph element related to the open connection. An objective function associated with system boundary is used to find optimal value of a design variable (evaluation criteria concerning the user requirements such as power consumption).

The components of meta-model are then represented in first order logic along with constraints for building a structurally valid topology so that there should be no unconnected ports in the design, no of connections is according to port fan-out or fan-in and only ports of compatible types are connected together etc. These constraints also ensure that the generated concept design is transformed into a valid partial simulation model using bond graph concepts.

Problem of generating alternatives is defined in terms of scope i.e. combination of particular number of elements categorised based on their port configuration (no of types of ports associated to an element) and total number of ports allowed to be used in solution generation. This problem definition of scope and first order representation of meta-model is transformed into binary satisfaction problem on which a SAT-solver is used to find Boolean solution which is then transformed back into original conceptual model representation of graph. Checks to determine whether the generated models are valid (in terms of signal and energy connections consistency) are done by checking the connections of energy/signal sink and sources with corresponding signal and energy I/O ports of the rest of the model.

Constraint satisfaction algorithm is then used to find variable value assignments for the design variables (e.g. transmission ratio) associated to the input-output relations of the elements by taking values associated to the I/O variables of system boundary (i.e. input/output profiles). The attributes of the elements are linked together according to the constraints specifying valid variable bindings (the variables at input ports of system boundary binds to variables at input port of system elements according to I/O relation). The constraints also specify permitted max-min ranges of effort-flow variable values.

For evaluation of concepts, a set of variable assignments are then chosen randomly using simulated annealing over the assignments generated by solving constraint satisfaction problem. A fixed value from design variable's value range is determined by random assignments and updated so as the effort-flow variable values of the output and intermediate ports (determined by input-output relation of the components) stays inside the given ranges. The criteria used as performance index is used to calculate the objective for that particular set of assignments. The next set of assignments is generated so as to minimise the objective function. The concept with lowest objective cost is then selected. A simulation model is then manually constructed by linking the partial simulation models associated to generated concepts and parameterised with chosen variable assignments.

It is demonstrated that by automatically generating design concepts wider solution space can be explored which also enables generation of innovative concepts, however this approach also generates unnecessary configurations because there is no check applied on functionality of the topology apart from structural check as it only uses structural elements. However use of functional or conceptual architecture can enable functionality checks and a more diverse range of physical architectures can be generated by following function allocation approach. This approach is more suitable for generation of parametrised simulation models for behaviour analysis of various types because of the modular nature of the partial simulation models associated to each structural element.

Engineering Design and Computing Laboratory from Swiss Federal Institute of Technology in Zurich (ETH Zurich) have pursued various research efforts to automate design synthesis in conceptual design phase for reducing effort in generation and evaluation of design alternatives. They have developed model libraries [25] where generic elements belonging to a product family can be stored to build domain independent structural models. This library contains primitive functions, function flow (port) types, structural components and structural connections. The connections are represented by blocks to specify components and their interfaces or ports to which they can be connected. Once a black box function model, "depicted as call behaviour actions", is created and the port attributes (flow type and direction) are

specified, the library then can be searched for physical components entailing the object flow ports with same attributes to build a physical structure model using internal block diagram.

Another effort [26] extends this approach by providing behaviour components in addition to function and structural components which can be transformed into simulation models.

2.3.2. Design Configuration

Efforts to automate design has been well underway from 1980's where rule based expert system have been used in expert systems like VT [19]. This automated system uses "propose and revise" method with forward and backward chaining to configure an elevator design which satisfies the customer, installation and regulatory requirements by using domain specific knowledge and domain independent problem strategies. These strategies includes proposing a part or component by comparing requirement parameters with its attribute values and fetching constraints over those attributes and requirement parameters. The unknown attributes values are estimated (using heuristics) and are corrected by searching for components with those attributes to satisfy the constraints related to them. Violation of constraints, when satisfying other constraint, is dealt by applying fixes (i.e. changing parameter value, selecting other parts etc.) ordered according to preferences while using look ahead rules to test the effect of changes induced by fixes on other constraints. The rule application reduces number of constraint violation by backtracking to the decisions stored in a dependency network, build during the forward chaining phase, to find further constraint violations due to applied changes. This design system is an ad-hoc approach for reasons such as rules, used in knowledge acquisition (during user requirements elicitation), are based on a pre-planned questionnaire for a predictable and static type of design.

NETSYN [59] estimates values of design properties by using probability estimation function implemented by a neural network which is trained through the use of partially completed designs, with known and unknown design property values, which are themselves randomly generated using rules constraining the range of values of the design properties. This approach is similar to above approach as it also proposes estimated values of unknown properties by applying probabilistic reasoning over other known and unknown property values which acts to limit the design space exploration. Author claims that such type of system representation and reasoning is useful in domains with weak domain theories e.g. behaviour of whole computer cannot be modelled in terms of input and output mathematical relationship.

[18] has followed a recursive case based reasoning (RCBR) process to perform system configuration, as part of system engineering development process, according to uncertain and inexact user requirements. It analyses the requirements to decompose them into sub-

requirements and then retrieve compatible solutions corresponding to those sub-requirements. In instances of retrieval failure that particular sub-requirement is further decomposed and same reasoning is applied again making the process recursive. Finally, the sub-solutions are integrated to create full solution. The case base is composed of hierarchical cases each of which is composed of requirements and their compatible solutions pertaining to a particular system along with similarity measures associated to those cases (if not present, designer preferences are used).

The requirement are modelled using flexible constraints which are used to express preferences and the requirement uncertainties. Compatibility measures are used to find appropriate values associated with stored solutions to satisfy flexible constraints. The stored requirement and solutions are represented in form of concepts represented in ontology which provide clear semantics and common terminology. These concepts helps to guide the designer by transforming requirements into descriptions of system that need to be designed and by providing knowledge of allowable changes to object properties. Retrieval of solution concepts based on given requirement concepts is performed by using similarity measures which entails user preferences. These concepts are composed of model of conceptual variables (entailing description of concept), model of domain (value that can be assigned to a variable) and model of conceptual constraints (constraining the constituent variables or constraints in which constituent variable takes part). Each solution concept is also constrained to a requirement concept, the solution variables can contain derived or added requirement variable copies similarly solution constraints can contained derived or added requirement as well derived solution constraint copies. The model of concept of a solution forms a generalisation/parent relation with requirement concept and with other high level solution concepts, therefore helps designer to define and instantiate constraints, variables and values of solutions.

2.4. Supervised and Unsupervised Machine Learning

Machine learning now a days is predominantly seen as estimation of a function mapping input to output for different type of tasks such as classification, regression, anomaly detection, data imputation etc. Machine learning consists of two phases[7] called training and testing. Main goal of machine learning is to reduce the generalisation error or test error which gives us indication of how well the model has been able to learn the relation between unseen (data that is not used as examples for learning) inputs and outputs e.g. an least square best fit line only passes through or near to known (x, y) values but we can infer new values of y using new x values near to

[7] Also sometime another phase after or during training phase is used to optimise hyper-parameters (parameters which are not updated with optimisation procedure) using a validation data set

known x values on the line. The data is divided into different ratios required for different phases and is normalised to same scale (either 0 to 1 or -1 to 1) to bring features or different dimensions (which may have different measurement units) of example to same reference scale. The training data is used to update the model parameter using an optimisation algorithm whereas the test data is used to calculate the generalisation error.

Different training methods exists to train the parameters of the selected model (encapsulating hypothesis functions space or potential input to output functions that can match actual function to be estimated) depending on whether analytical form of the model exists or not. If the analytic form exists e.g. like in linear regression of the form given in Equation 2-1, then methods such as gradient descent can be applied to learn its parameters by feeding full training set at a time. Model parameter training consists of minimising a given cost or loss function (like in Equation 2-2) generally given as mean squared error between estimated and actual output for real values.

However, now a day's data sets ranging in magnitude of millions of data points (or samples) with high dimensionality (e.g. 64 × 64 image with 3 RGB colour channels has 64 × 64 × 3 dimensions) exist, therefore it is impractical to feed all training data set once at a time due to memory restrictions, hence stochastic gradient descent optimisation algorithm (can only be used on cost functions which sums over the individual sample costs and therefore can be averaged over full training data set) is now used where random subsets of training data (called mini batches) is fed during each training iteration to estimate the gradient of the cost function.

$$\hat{y} = w^T x$$

Equation 2-1

Where $\hat{\boldsymbol{y}}$ is the estimated output, w^T is the weight vector (also called model parameter) to be learnt and x is input with different features or dimensions of the x is weighed by an independent weight.

$$L = \frac{1}{N}\sum_{i=0}^{N}(\hat{y}_i - y_i)^2$$

Equation 2-2

Neural Networks (NN) are a class of machine learning models capable of learning non-linear functions by using a composition of linear functions (connected through non-linear activation function[8]) where each (apart from last function delivering the output) function's output is fed

[8] The activation functions (e.g. a softplus function $\log(1+e^{x+})$) usually contains no learnable parameter hence are not counted as layer of NN

to another function forming a chain of functions e.g. $f(x) = f^3(f^2(f^1(x)))$. Each function then forms a layer of a NN and intermediate functions are called hidden layers. Because of using multiple stacked layer in neural networks this machine learning approach is commonly referred to as "Deep Learning". This composition of function forms interconnected layers of parameters capable of representing any function but not guaranteed to generalise or learn that function [60, pp. 198-199].

Linear regression is form of supervised machine learning in which the actual target is known whereas in un-supervised machine learning actual target is to learn or approximate the data distribution over the input i.e. p(x).

2.4.1. Feature Representation Learning with Auto-encoders

Auto encoders summarizes high dimensional input into low dimensional space or representation (also called "code" or "latent space" or "embedding" *z*) [61] whose each feature or dimension corresponds to a distinct feature of input data having high variance. This compression brings data with similar features together (i.e. having minimum distance across each dimension) e.g. when embedding natural language the embedding's of sentences containing words belonging to pet category e.g. dog or cat, will be closer to each other [62].

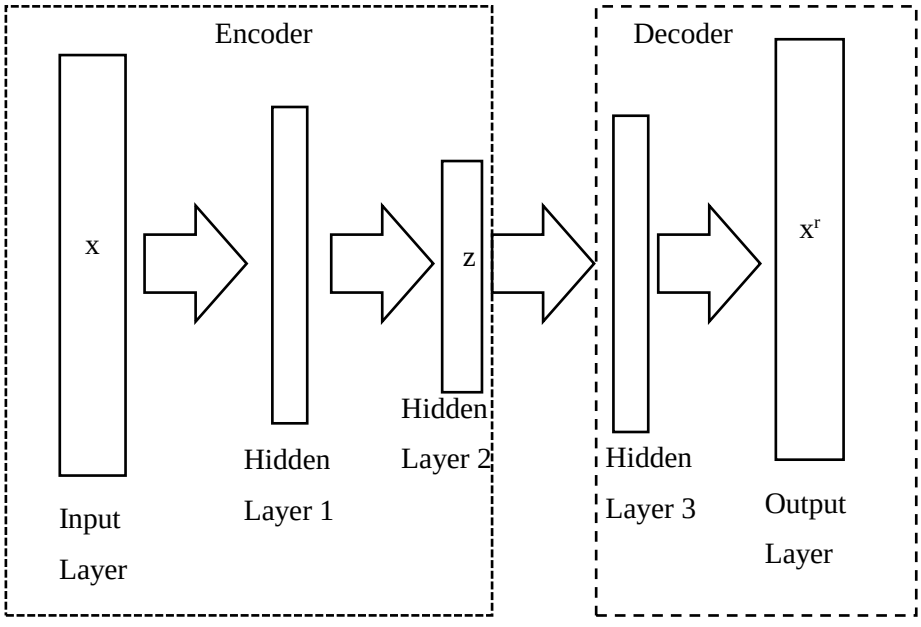


Figure 2-6 Example Auto-encoder architecture with connections represented by arrows

Auto-encoder Neural Network is an example of un-supervised[9] learning which uses input as its output i.e. it tries to reconstruct the input (x^r) as shown in Figure 2-6. The goal of auto-encoder is to learn independent underlying factors of variation which has given rise to the data. These factors are represented by using an intermediate layer output or representation for each input and the goal is to represent different factor of variation (shared by all inputs) along different dimension of the representation.

VAE or Variational[10] Auto-encoder [63] relies on capturing most salient (distinct/non correlated) features or dominant factor of variations in data by encoding the latent space in such a way that the output which closely matches with the input is assigned high probability given the latent space that is constructed. Figure 2-7 shows example of facial expression image generation where horizontal axis capture the rotation of head and vertical axis capture the change in expression. The cost function used in VAE consists of reconstruction error or negative log likelihood (mean squared error for real valued data or Bernoulli loss for binary data) and regularisation term which measures the error between the assumed distribution or model prior over the latent variables p(z) and estimated approximate posterior distribution q(z|x) by the VAE. A trade-off exists between how well one wants output to match the input (reconstruction loss) and how well one wants the latent space to generalise w.r.t input (so that model does not over fit the training set) or how much one wants the reconstruction to rely on latent space that is a random sample with minimum regard to data distribution of input x? Generally the reconstruction term in VAE dominates especially in case of small data sets, models with limited capacities and when dimensionality of input is very high compared to embedding z dimension [64]. The bottle neck in VAE is the assumption of assuming a particular type of distribution p(z) for latent features to which the learnt approximate latent posterior distribution q(z|x) tries to match e.g. if p(z) is assumed to be uniform then without taking x into account different values can be sampled from q(z|x) to be as random as possible or on the other hand if p(z) assumed to be Gaussian then same z can be sampled from q(z|x) for many different data samples representing x because all samples of x are summed over or averaged (using $p(x^i)$)to get z. Increasing the regularisation term's coefficient in VAE loss helps to generalise better over small set of examples, but it decouples input from q(z|x) further as pointed out by authors in [64] and they devised new formulation of loss term for VAE called "info-VAE" and if MMD (Maximum Mean Discrepancy) measure is used then its called MMD-VAE which

[9] Auto-encoder training requires use of input as output target but the actual output target lies in the latent space of achieving least distance between similar examples sharing a common factor of variation.

[10] Based on converting probability distribution inference problem into optimisation problem where distance between parameterised posterior distribution (over intermediate output or latent factors given input) and a target distribution (belonging to a family of distribution e.g. Gaussian) is minimized.

allows use of high coefficient values on new regularisation term without any side-effects. MMD measure compares similarity between two distributions based on their moments inferred from the samples of distributions, therefore it does not try to compare q(z|x) and p(z) based on every sample but based on their expectations, this forces the use of input samples when inferring z from q(z|x).

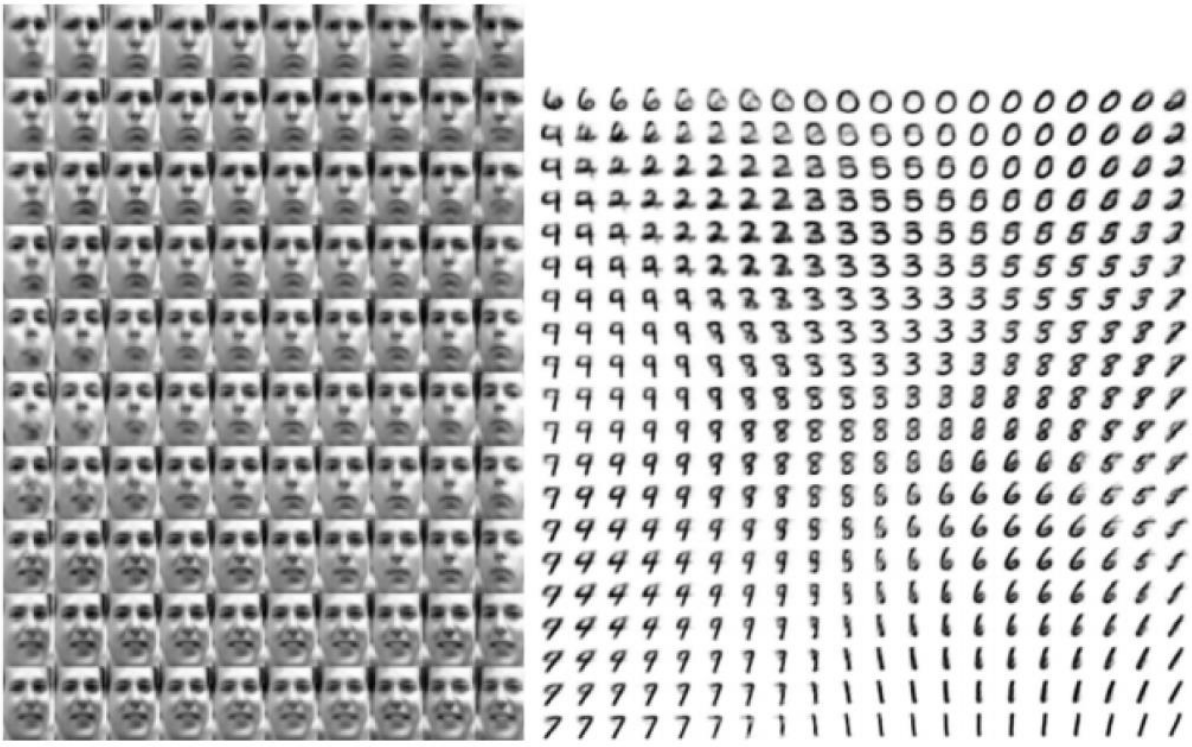


Figure 2-7 Variation in a particular feature in the facial expression (on left) and hand written digit (on right) images generated by varying z in each of the 2 dimensions of the embedding space learnt by a VAE [63]

Chapter 3 Problem Formulation of System Model Development

This chapter identifies some of the key system modelling outputs required to construct a valid system model w.r.t available designs and different type of constraints. These outputs are then formalised in terms of the concepts involved in each output and their relation to concepts of other outputs by focusing on parameter and property value spaces and formulation's compatibility with application of logic satisfiability, constraint satisfaction and machine learning techniques. Based on this problem definition a conceptual framework is proposed which addresses part of the problem defined through use of AI technologies.

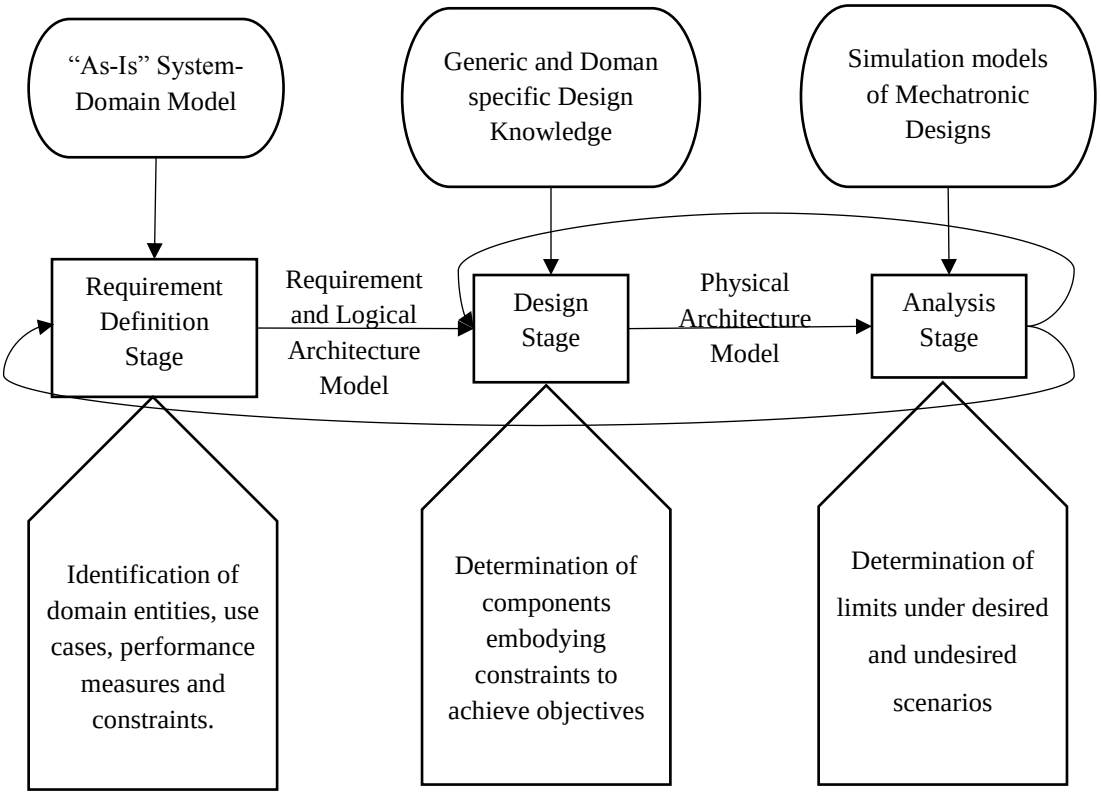


Figure 3-1 Relationship between different system modelling tasks, external inputs and modelling outputs

This chapter formulates problem definition of system modelling based on the MBSE methodologies review in section 2.1.1. and key MBSE tasks identified for computational support in section 2.1.2. We can summarize the main outputs of each stage of system modelling shown in Figure 3-1 as following,

1. Requirement definition stage:
   i. Identification of external domain entities and existing system interfaces
   ii. Identification of high level system functions and use cases utilising those functions w.r.t user interaction.
   iii. Identification of parameters characterising user interaction in use cases.

iv. Identification of parameters and properties of external entities w.r.t high level functions.
v. Identification of system design constraints and performance measures.
vi. Identification of low level functions and interaction between them (in form of function chains) based on each use case for different user command and environment parameter value combination.

2. Design stage:
   i. Grouping and decomposition of logical/physical subsystems so as to match their functionality with functionality delivered by available components.
   ii. Selection of physical components for function realisation and deduction of correct topological arrangement for them.
   iii. Elaboration of subsystem interfaces and therefore interactions between the components using component specific operations/methods and properties.
3. Analysis stage:
   i. Identification of constraints from integrated sub-systems on performance measures in different scenarios under different modelled and un-modelled interactions.
   ii. Identification of undesired phenomenon affecting performance measures due to interaction between component properties under unconsidered input parameter values.
   iii. Determination of whether required functionality can be achieved by existing designs.
   iv. Identification of limits over characteristics of system's expected function behaviour based on various types of constraints.

3.1. Illustrative Example

In this section, example of development of internal lighting system model using set formulation will be demonstrated. Set formulation is used because items in set are not required to follow a particular order, items can be dynamically added or removed without affecting constraints between existing items and sets can be transformed into other formulations such as graphs by addition of directional dependencies. In following sections, set formulation is used in formalising the concepts involved in the output models of different modelling stages.

Suppose we are developing model for Household and Automotive SADs (System Application Domains).

These domains could be specified further into e.g. Household = {Indoor, Outdoor, Adult, Children}, Automotive = {Haulage, Private, Construction, Maintenance}, if needed. From here on "Type: Subtype" convention will be used to represent subtype relation.

If considered each of such sub-domains can have some variations on properties of available components e.g. switch or dial types, dial rotation range, toggle button stiffness range, bulb luminosity value range, bulb type etc. In addition, there could be additional specific requirements for performance, power efficiency, safety and robustness pertaining to the SAD.

Also suppose a repository of simulation models modelling light circuits exists in different domains with variety of voltage/current altering components, bulb powers, input source types (circuit can be driven by a current source for some specific type of bulb) etc.

System Model can be defined by defining the Entity, Requirement, Logical and Physical architecture models. These models and key concepts used in these models are defined in next sections.

Entity model

The Entity model captures by numbers and types of entities present in the "as-is" system & domain. These entities are used in the succeeding models.

Entity Model = {(System1, "SAD", Household), (Human, "type", User: Household), (Ambient light, "type", Room Environment: Household), (System, "contains", Lf_i: Light fixtures), (Lf_i: Light fixtures, "interfaces", Room Environment: Household)} $i \in \{1 \ldots 10\}$

(System1, "SAD", Household) is triple formed of subject, predicate and object. Where classes "Household", "User" and "Room Environment" are assumed to be defined in terms of their number, type and permissible value ranges of attributes (properties and parameters) as well as relations between the attributes.

Functional Requirements

Fr1 = {(System1, "perform", Control Light Intensity), (Human, "hasParam", inp1: Input: HMI), ($Lf1_i$, "perform", output light: Provide Light[11]), ($Lf1_i$, "hasParam", Lfp: Lumen), (Ambient Light, "hasParam", Lfa: Lumen)} for $i \in \{1 \ldots 10\}$,

Fr2 = {(System1, "perform", Selective Lighting Control), ($Lf1_i$, "perform", Change Light Intensity), ($Lf1_i$, "hasParam", Lfp: Lumen), (Human, "hasParam", inp2: Input: HMI: Household)} for $i \in \{1,10\}$

Two function, Fr1 and Fr2, requirements based on two use cases of controlling a selected light fixture and selecting a particular light fixture to control has been defined using functions such

[11] "Provide light" can be decomposed into convert electrical to light and distribute light functions.

as Control Light Intensity, Selective Lighting Control etc., with their corresponding input and output parameters such as Inp1, Lfp, Lfa etc.

Operational Requirements

Op1 = {(Human, “hasParam”, inp1: Input: HMI: Household), (inp1: input: HMI: Household, “hasValue”, Pint1: Integer), (Pint1, ≥, 0), (Pint1, ≤, 10), (Ambient Light, “hasParam”, Lfa: Lumen), (Lfa, “hasValue”, Amb1: Integer), (Amb1, ≥, 0), (Amb1, ≤, 100)}

All the values assigned to parameters should be according to the Household domain i.e. the value range given should intersect the Household SAD values or more strictly should be inside the boundaries defined by Household SAD.

Similarly a scenario for the 2nd functional requirement Fr2 can be created by assigning value ranges to parameters.

Static logical architecture

The first level of function decomposition can be defined as following,

(Control Light Intensity, “hasSubFunction”, Measure Light Intensity: Sense), (Control Light Intensity, “hasSubFunction”, Change Light Intensity: Act), (Control Light Intensity, “hasSubFunction”, Produce Light: Electrical-to-Light) (Control Light Intensity, “hasSubFunction”, Compare: Signal Compare)

(Measure Light Intensity, “hasParam”, Ambient Light: Lumen), (Change Amplitude, “hasParam”, E1: Electrical: Household) (Produce Light, “hasParam”, Lm1: Lumen) (Produce Light, “hasParam”, E1: Electrical)

(Selective Lighting Control, “hasSubFunction”, Convert Index to Position), (Selective Lighting Control, “hasSubFunction”, Read Input Index)

(Read Input Index, “hasSubFunction”, Take Input) (Read Input Index, “hasSubFunction”, Store Input)

(Read Input Index, “hasParam”, inp2: Input: HMI: Household) (Take Input, “hasParam”, read value) (Store Input, “hasParam”, store value)

(Convert Index to Position, “hasParam”, inp2: Input: HMI: Household) (Convert Index to Position, “hasParam”, position1: De-multiplexer)

Behaviour logic architecture

Next the decomposed functions can be further specified in terms of their expected behaviour and interactions between them based on usage of each other's parameters. Then functional sequences can be constructed according to use case scenarios in function and operational requirements

Take Input = [if read value(t) = 0 then read value(t) = inp2(t)] ∀ t ∈ ON Duration, where 0 is ideal state value.

Store Input = [if store value(t) != read value(t) then store value(t) = read value(t)] ∀ t ∈ ON Duration

Convert Index to Position = [position1 = look up function(store value(t))]

Above two functions uses logical conditions which can lead to different behaviour traces based on conditions are satisfied or not. However for sake of representation, default values and alternative values are not shown in case of conditions being not satisfied.

Measure Light Intensity = [sample1(t) = Ambient Light (t)] ∀ t ∈ ON Duration

Compare = [comp1(t) = absolute(sample1(t) – inp1(t))] ∀ t ∈ ON Duration

Change Amplitude = [for each i ∈ {0..10} ∃ one c ∈ {0…100}: if position1 == i then c>0 else c=0 and E1 = c × comp1(t)] ∀ t ∈ ON Duration

Produce Light = [for some η ∈ {0…100}, Lm1 = η E1]

The "Change amplitude" has multiple sub-behaviours and only one which can be activated at a time based on "position1" parameter of "Convert Index to Position" function.

All the physical parameters exchanged between functions e.g. E1, should always has its value bounded by the Household SAD.

Sub-System or Component allocation

The low level functions can be assigned one or more generic subsystems that can achieve their specified expected functionality by matching number[12] and type of parameters as an initial search condition. E.g. the "Measure Light Intensity, Compare, Change Light Intensity and Produce Light" functions can all be achieved by one physical subsystem made either of a

[12] Only subset of parameters of existing components can be matched as described in section 3.5.

mixture of digital (microcontroller etc.) and analogue electronics or just by analogue electronic circuits.

- The condition of allocation is that the inter-dependency of parameters of functions is still respected.
- All the low level function's parameters are classified further (w.r.t Engineering Domain of selected solution e.g. Mechanical, Electrical etc.) and not abstracted such that their value domains are constraint further but still reside under SAD.

e.g. (Microcontroller, "realises", Compare) (Microcontroller, "hasParam", d1: Digital Voltage) (Microcontroller, "hasParam", inp1: Internal Variable) (Microcontroller, "hasProp", res1: AC-DC resolution) (Microcontroller, "operatesIn", op1: Votlage) (op1, "hasValue", N1: Nominal Voltage Domain)

(Current Source: Transistor Current Source Circuit, "realises", Change amplitude) (Current Source, "hasParam", I1: Collector Current) (Current Source, "operatesIn", op2: Base Current) (op2, "hasValue", Lin1: Linear Transistor Range and Ln2: Cut Off Range)

The selected generic components or physical subsystems (if more than one alternative exists) then can be further pruned in terms of restricting their property value ranges by specialising the SAD further i.e. Household SAD can be further specified by residential types e.g. Flat: Household SAD, House: Household SAD etc. The specialisation of domain should be such that the system design or global property constraint requirements (e.g. maximum power usage, Household voltage etc.) can be satisfied.

Analysis of compatibility between allocated designs and system architecture and requirement model

The descriptive models of components or physical subsystems thus retrieved needs to have corresponding instantiable simulation model (i.e. model that can be validly constructed e.g. power source to sink path should be complete, model should allow the usage of friction, resistance or appropriate damping to keep derivatives of all signals finite) produce all the required behaviour to meet the performance measures while staying within the constraints of performance measures. A design can be called functionally verified if one valid instantiable model from simulation model repository can be selected by matching parameter types of input (source or any intermediate parameter in simulation model) and output (any intermediate parameter in simulation model) as well as parameter value domains with the designated input and output parameters of selected descriptive models. Similarly, if the matched parameters in the simulation model produces same behaviour as enumerated behaviour in logic function

(substituted by the descriptive design) under each use case scenario or system input then we can say that design has verified behaviour. This verification approach can therefore be seen as design selection process.

e.g. if a power transistor circuit exist with potential to operate in both linear and cut-off regions while using Household power supply (Change amplitude function's parameter E1 has domain Household) as well as can produce the linear relations in required range (based on selected values of inp1 and constant *c* in Change amplitude function) between parameter comp1 of compare function and collector current I1 then that circuit can be selected.

3.2. Formalisation of Concepts Common across Stages

Outputs of requirement definition, design and analysis stage can be represented by using concepts formalised in sections below, starting with concepts common to all stages.

Given observable vectors $\mathbf{E}(s, t) \in R^N$, $\mathbf{U}(t) \in R^K$ and $\mathbf{X}(t) \in R^M$ of corresponding environment (depends on space and time), user command and desired system parameters, respectively, for some use case scenarios along with measurable system properties set **Po**(s, t) from the system model without specific component property constraints. Where user properties can be included in the environment parameters. Design task is to determine required constraint functions C: $\mathbf{E} \times \mathbf{Po} \times \mathbf{U} \rightarrow \mathbf{X}$ realisable by known component phenomenon given system property constraints on component property values. Because input and desired parameter vectors are only given for some scenarios i.e. **E**(s, t) is not fully known a prior, Analysis task is also to identify those input parameter combinations which are not covered by the constraint functions. Let cardinality of $\mathbf{P_{Model}} = \{\mathbf{N}, \mathbf{K}, \mathbf{M}\}$[13] where $\mathbf{P_{Model}}$ is the set of all parameters in model i.e. from system and external entity parameter sets. The parameters exchanged between system and environment can be given as a set $\mathbf{P_{External}} = \{ \mathbf{P_{System}} \cap \mathbf{P_{Environment}} \}$

Definition of Entity:

Entity = {∀ E1, ∃ E2, E3, Re1, Re2, Re3, Pr1: Re1 ∈ contains Re2 ∈ interfaces Re3 ∈ hasParameter Pr1, Pr2 ∈ $\mathbf{P_{External}}$ E1, E2, E3 ∈ C (E1, Re1, E2) (E2, Re2, E3) (E2, Re3, Pr1) (E3, Re3, Pr2) Pr1 = Pr2}[14], where C is set of classes from system application domain and the equality in Pr1 = Pr2 is in terms of equivalency over the domain values of both parameters.

The model under development targets a particular domain usage e.g. automobile, household etc. Therefore, property constraints and operational requirements specifies required metric

[13] **N** are control inputs, **K** are disturbances inputs and **M** are controlled outputs

[14] These types of sets can be easily expressed using quantifiers, binary and unary predicates of first order logic formulas [47]

targets w.r.t to application domain. On the other hand, existing design solutions are also normally constructed with a particular application domain in mind. The system application domain (SAD) is defined and related to other SADs based on design's physical properties and input-output parameter characteristics. But different modelling tasks can benefit better from using one or the other characterisation criteria e.g. it is more beneficial to first select designs based only on the behaviour characteristics of the function being realised as this provides more solution options some of which may be somewhat violating domain specific operational environment and design constraints (SAD is discussed further in section 3.8.).

Each entity (including system) should be classified to a class from a set such as C = {C1, C2…Cn}$_{\text{Domain}}$ which can contain both system and environment types. Such a class set is determined from the SAD. Environment and user forms part of external entities i.e. specific type of entity class.

The environmental, user and system parameter as well as the property values belongs to particular set are determined by SAD (determined by design constraints pertaining to "as-is" System and Domain class) i.e. $D_E \subseteq R^N$ $D_U \subseteq R^K$ $D_X \subseteq R^M$, where all domains of **E**, **U** and **X** are determined from classification of system model to SAD. Let **I** = {D_E , D_U} and **O** = {D_X}.

A particular solution configuration and its behaviour satisfying the textual, behaviour and functional requirements is shown in Figure 3-2. However, same behaviour can also be obtained from use of hydraulic motors and internal combustion engines which however may not satisfy the design domain constraints (not stated here) of household domain where size and mass of design has to be lower than other domains such as automobile (car, truck etc.) domain.

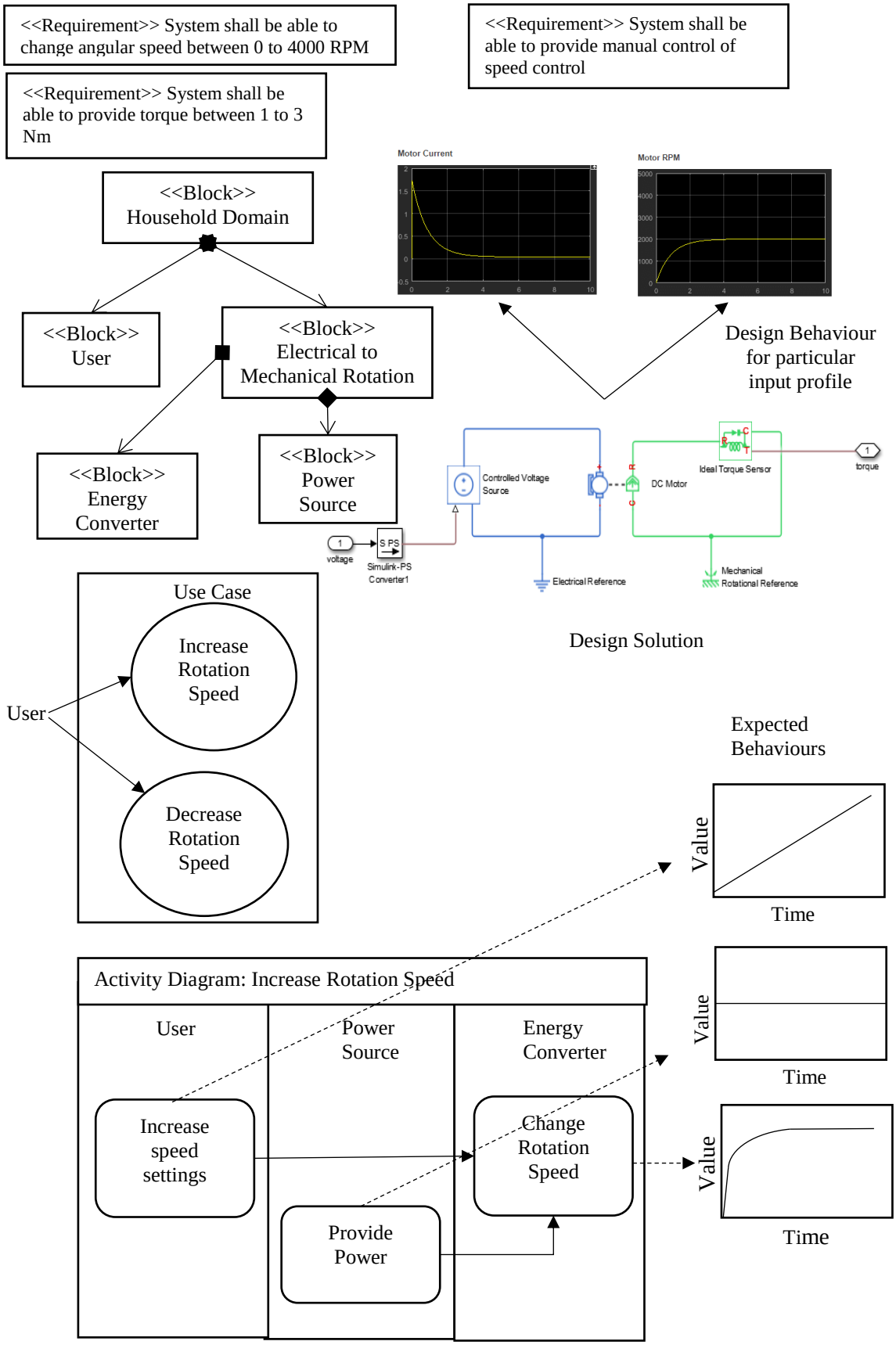


Figure 3-2 Simple system model relating requirements, function, Structure and behaviour using SysML like constructs

3.3. Formalisation of Key Concepts of Requirement Definition Stage

A requirement in general can be given as a set of triples {subject, predicate, object} that specify known attributes about each entity as well as known relation between entities.

Different types of system requirements can be represented in models but only following first 2 are considered for formalisation in this section whereas third one is the artefact that is produced as a result of system model development after allocation of physical subsystems to available physical components however is informed by global property design constraints and operational requirements mentioned in this section,

i. Functional requirements specifies what the system is expected to do (reduce vertical vibrations).
ii. Operational requirements are given in terms of constraints between non-functional requirements for different scenarios (e.g. reduce vertical vibration for undulations of x height at y speed).
iii. Design requirements specifications specifies required property values for decomposed sub-systems and are derived from satisfaction of operational requirements by the derived functional decomposition under given design constraints.

Fr1 = {(E1, Re1, C_s1), (E1, Re2, F1), (F1, Re3, Pr1)}, where Fr1 is one of the functional requirement, F1 is a function, Pr1 is a parameter, Cs1 is one of the class from SAD and Re can be domain independent or dependent relation (e.g. generalisation, decomposition, logical, equal to, less than, "hasParameter" or any relation defined as per SAD). These requirement can use pre-defined[15] entities, function, parameter and properties types belonging to the SAD. The high level functions used in requirements represents particular type of mapping between same or different type of system parameters.

The operational requirements are given in terms of constraints specifying non-functional requirements for different scenarios (e.g. reduce vertical vibration for undulations of x height at y speed). When defining the operational requirements it is essential to refer to the system boundary and the expected entities that will interact in different scenarios with the system based on Entity model.

[15] The definition can be given in a meta-model constructed using an ontology.

Extending the functional requirement definition, an operational requirement definition will also link different entities or their characteristics together for a specific state (or values) of parameters.

O1 = { (E1, Re1, C_s1), (E1, Re2, F1), (F1, Re3, Pr1), (E1, Re4, E2), (E2, Re2, F2), (F2, Re3, Pr2), (Pr1, Re5, Pr2), (Pr1, Re6, Va1), (Pr2, Re6, Va2), Va1 ∈ (x_1….x_n), Va2 ∈ (y_1….y_n)}. Re6 can be a relation such as "hasValue" to a set of type Value (e.g. Va1 or Va2) containing integer or real numbers. The values in Va1 and Va2 set can be defined based on either a mathematical relation or approximate range of values with bounds applied w.r.t SAD classification of the function and particular parameter type. The parameters referred in the operational requirement are assumed to be dependent i.e. one is input (e.g. power input, control input, disturbance etc.) and other is output (e.g. controlled output, by-product etc.). Use cases are more abstract type of operational requirements where parameters and constraints on their values are not specified but only external entities interaction with specific aspect of system's function(s) is captured e.g. a user (E2) drives a car (E1) and uses its control speed function (F1) to increase speed (F2 sub-function of F1 or F2 ⊆ F1).

Use of such set based definitions relates all the referenced information into a single context.

These requirements then leads to construction of logical architecture formed of interconnected functions or their sub-functions.

Apart from above requirements, system design constraints are also given in requirements therefore we can define them as *global property constraints* constraining all properties of same type or sub type (e.g. limit on system mass constraints all components mass). Formally, they also form a set by using mathematical relations such as {(Po1, Re1, C1)}. Based on this set other such sets can then be derived during design stage forming part of design specification such as {(Po2, Re1, C2), (Po3, Re1, C3)} where Po1 is system property while Po2 and Po3 are component properties under constraint of constants C2 and C3. C2 and C3 will then implicitly be related to C1 by virtue of some physical law induced by design structure. The design constraint applied on components can also refer to sub-type of Po1 property type in case where more detailed design specifications are required e.g. if Po1 ⊆ Co1, Co2 ⊆ Co1 and Po2 ⊆ Co2 where Co1 is a type of property e.g. mass and Co2 can be one of component of mass around a particular axis in system's local reference frame.

The functions referenced in the requirements are conceptual functions which can be decomposed into sub-functions that can be used to define the expected behaviour in more detail and also act as placeholders for available designs or components.

3.4. Logical Architecture Construction w.r.t Pre-Defined Use Case Scenarios

A logic function is composed of specific type of parameters and relations between them such that the relation can be formally defined w.r.t space and/or time, over a bounded set of parameter values e.g. a function called "Control" controlling a specific type of entity using its associated parameters can be specified as a set $\{\forall\ t: v = c\} = \{(v1,t1,c1), (v2,t2,c2)\dots\}$, where t is time and v is the controlled parameter set to reference value c and at each time step t values (c2,c1 etc) of parameters stays equal. A logic function can contain a set of ideal behaviours or set of mathematical relations and only one of which is activated at a time to produces non zero output parameter value set based on input value range. A component or subsystem can then be selected (to execute the function(s)) which exhibit all such behaviours in at least one of its state. A combination of all active functions are represented by desired excitation of these logic functions with different input types or triggering of particular guards in logic conditions.

A conceptual or logical function is formally defined as a set of finite enumerations over one or more parameter's values e.g. $F1 = \{(P1, P2,\dots Pn) \subseteq P\}$ i.e. set of tuples of parameters with domain value set bounded by SAD. Let P be the set of all parameters used in the logical architecture and $P \subset \mathbf{P_{Model}}$. Sub-function of a function i.e. $F1 = \{F1a, F1b. F1j\}$ $j \in \mathbf{J} \subset \mathbb{Z}$ can be defined as a set of subset of function's parameter and/or subset of domain values of those parameter. These functions contains internal parameters $\mathbf{P_{in}} \subset P$ such that they have domain $D_P \in \mathbf{I} \cup \mathbf{O}$ i.e. some of them must have same units and overlapping value set with input-output parameter value set $\mathbf{I} \cup \mathbf{O}$. In other words, logic parameters should covers full or partial domains of system's inputs and output parameter values.

Here space and time can also be used as a parameter e.g. control speed = {(Ref1, t1, Act1), (Ref2, t2, Act2)…. } where Ref is reference parameter value and Act is actual speed value and the relation between two is defined w.r.t time.

A set of constraints (expected functions) in form of connected L logic functions (through their conceptual input and output port) forms a static logical architecture such that some of these functions consists of system's input and output parameters, where $L = \{F1, F2 \dots Fn\}$ with $n \in \mathbf{N} \subset \mathbb{Z}$ a finite integer.

The choice of constraint function types (or the range of input/output values and form of mathematical relation represented by them) also have to be restricted by the respective application domain because components or subsystems (having capability to go into particular

states[16]) required to realise those constraints are subjective to combination of input values (determined by application domain) exciting particular states of components.

A system can have different uses or external interactions in different scenarios because of which it can execute a different function sequence (thus a different behaviour) in each scenario causing system to have different operational states. Therefore a behavioural logical architecture can be produced by adding logic conditions on input or output parameters of functions to represent different outcomes or scenarios of a use case. Different behaviour traces can then be produced for different use case scenarios (having different set of input values as per operational requirements) activating particular (sub-)behaviour and/or selecting particular logical branches at each logic condition evaluation.

The set of achievable expected system states should then be equal to all possible behaviour traces.

3.5. Representing Design Solutions at High Level

Main output of the system modelling activity is a consistent design specification (containing system's I/O specification and sizing[17] information). The design specification specifies required property values based on parameter value domains of decomposed functions satisfying operational requirements under given design constraints. The design specification is generally derived either by calculating the property value domains based on the parameter domains (considering the worst case values) of each function or based on the simulation of ideal or abstract mathematical models representing prospective components that can realise the sub-functions. We will consider the later approach because the expectation is that system model developer is not expert in every engineering domain.

Design space contains feasible solutions or realisable constraints in form of combination of components or physical (not logical) subsystems (representing available technologies) and their states. The aim is to construct physical architecture model by selecting components or physical subsystems from available design space such that a consistent design specification from it can be derived.

A component is an instantiation of physical law in form of relation between properties and parameters with closed value sets (e.g. limited force for limited size) and having interaction dimension defined by the subset of parameters. Therefore a component can be defined as a set

[16] State refers to a particular partition of parameter value space and a component can show a particular type of I/O behaviour for particular range of input parameters e.g. a transistor can go into linear or saturation based on base current.
[17] Design sizing refers to the range of physical design property value

of sets over properties and parameters for each value of property(s) e.g. for two parameter Pr1 and Pr2 and one property Po1, a set containing two sets can be construed for each value of Po1 i.e. { {(Pr1_1, Po1_1, Pr2_1), (Pr1_2, Po1_1, Pr2_2), …. (Pr1_n, Po1_1, Pr2_n)} {(Pr1_1, Po1_2, Pr2_1), (Pr1_2, Po1_2, Pr2_2) …. (Pr1_j, Po1_2, Pr2_j)} } n ∈ **N** , j ∈ **J** with **N**, **J** ⊆ ℤ, where **N** and **J** is selected such that characteristic component behaviour can be represented. Here we have used component's property to indicate change in component's mode of operation. We assume that each parameter is dependent on other parameter i.e. not considering a connection between them and assuming bi-causality. This assumption then determines number of subsets along with combination of number of properties, property values and parameters. The change in the property value can result either due to the way component is used in relation to other components in the design, input parameter value and/or external parasitic effects.

Set of probable design solutions or physical architectures comprised of different components configurations should be selected by matching individual (or group of) expected functions/sub-functions in static logical architecture with number and types of parameters as well as by matching characteristic functions of components or of subsystems in different operation modes or component states with behaviour traces from behaviour logic architecture. Only the design solution which subsume or generalises over all behaviour traces should be selected.

Each physical architecture then can be represented as combination of components in form of a set, such as **Ds** = {C1, C2….Cd} d ∈ ℤ, which transforms **I** -> D_P -> **O** for each $\mathbf{P_{in}} \subseteq \mathbf{P_{rc}}$, where $\mathbf{P_{rc}}$ is parameter set of set **Ds** with domain D_{pc} such that $D_P \subseteq D_{pc}$.

This set of selected alternative designs or architectures can then be reduced by only selecting those which can satisfy end to end behaviour relations between system's input to output parameters such that parameter values always stay in SAD for each behaviour path (resulted due to activation of sub functions and branching from logic conditions) defined in the high level use cases of logical behaviour architecture.

In our case, designs are represented by simulation models. Therefore task is to determine set (**SIM**) of simulation models (representing realisable design constraints embodied in components) for all intended behaviours defined in use case scenarios which are compatible (e.g. matching parameter type, parameter directionality for connectivity etc.) with each other, also contains same type as well as equal to or greater than number of parameters as P. The union of domains of all simulation model's parameters should cover the **I** ∪ **O** set. The mathematical functions embodied by simulation models between input-output parameter pairs (p_i, p_j) in P should be same (or more general than) as functions in L (i.e. the selected design should underpin

or satisfy at least one of the behaviour or mapping in the logic function that can be activated based on inputs from other functions or from system input).

3.6. Analyses of Selected Design Solutions at System Level

The final system property and parameter values are determined from selected component's constraint functions constraining their and other components parameters and properties. The components configuration applies constraint function (e.g. tension force applied by screw between surfaces) which interact with external forces due to environment and user command parameters to achieve desired system parameter values in different scenarios. These constraint functions can themselves change depending on the component state change due to change in component property values (i.e. max/min tension achievable by component affected by change in its stiffness due to temperature). Design property constraint requirements applies limits on system property values which are then transformed into limits over component properties which then affects the range of their constraint functions (e.g. gear reduction ratio depends on gear size which is constraint by the system space constraint therefore only limited range of output torque is possible). This cascade of effects on the overall system represented by selected physical components is shown in Figure 3-3.

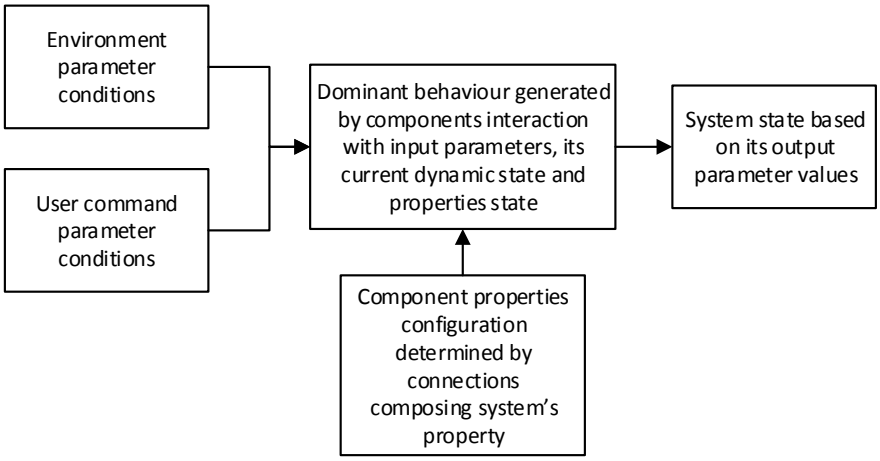


Figure 3-3 Effects of selected components and external parameter on overall system behaviour.

Model analysis is therefore task of ensuring presence and absence of correct constraint functions over the required system parameter, user command and environment parameter ranges.

This stage of system modelling ensures correct integration of components or physical subsystems. This stage can be catered in similar way to selection of designs but here selected designs or components are used to find value sets of systems parameter which are outside the

value constraints defined in operational requirements, instead of other way round. Analysis is normally used for verification and validation (V&V) of the system but here we are only dealing with some abstraction of the system captured in models. Therefore the analysis are conducted w.r.t knowledge available in form of domain specific constraints (as in domain specific meta model in section 4.2.1. e.g. power sink and source type of components should not be connected directly) and simulation environment. V&V in context of model analysis means that verification is performed to verify correctness of the descriptive physical architecture model as well as its corresponding simulation model whereas validation consists of finding caveats in the requirement and logical architecture model surfaced during extended[18] analysis of physical architecture.

The verification is divided into two parts namely functional and dynamic behaviour verification. The functionality of the system is verified by determining if selected components or subsystems are connected in such a way that they achieve the required mapping between system and logic function parameters while substituting all logic function constraints and residing at correct level of abstraction (e.g. the value domains of electrical parameters of "convert AC to DC" function should be same i.e. American Household or European Household etc.). The behaviour verification is performed by determining similarity between required system and simulated input to output behaviour under operational environment's physical constraints as shown in Figure 3-4. It is required that selected design should cover all the input to output behaviour space i.e. generate required outputs for given inputs.

[18] Beyond the expected behaviours and limits on parameters in operational requirements

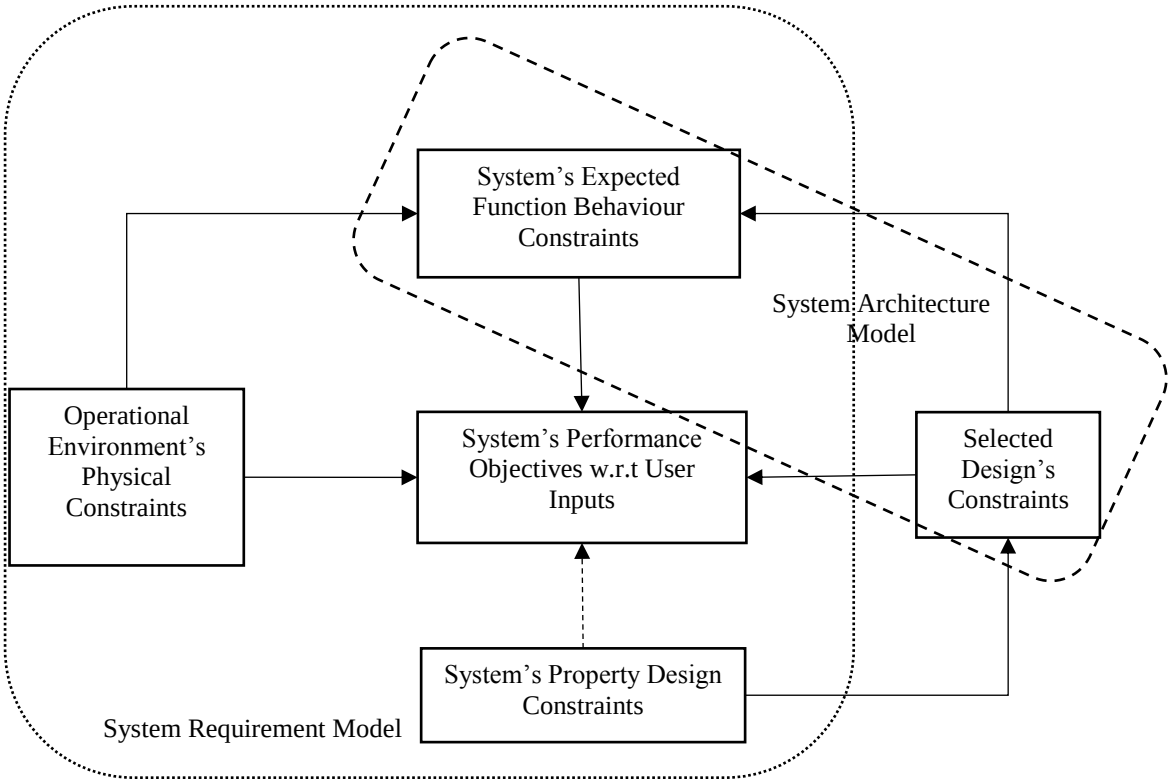


Figure 3-4 Arrows represent restrictions applied (directly – solid line and indirectly – dashed line) by one part of the model on the other.

The system can usually undergo new combination of input values which are not accounted in operational requirements and executes function behaviours orthogonal to behaviour traces of logical architecture, the validation of model identifies such inputs and resulting behaviours. The components selected during the design process can express new behaviour states when excited by unknown input values, to induce undesired behaviours adversely affecting the expected system behaviour or limiting the achievement of full value range of expected system behaviour. The activation of new component modes or states happens as a virtue of combination of chosen component arrangement (to satisfy only required logic constraint functions) as well as component property and system input values. The chosen configuration of components or design solutions can also induce undesired coupling between expected functions leading to undesired system behaviours. Figure 3-4 illustrates effects of various constraints on performance objectives defined on system's output behaviour.

For example if System $\mathcal{S}$ formed of $\mathbf{N} \times \mathbf{K}$ inputs (control + disturbances) and $\mathbf{M}$ (controlled) outputs with expected values range given by system application domain and external entity states (for each interaction/input profile) for respective system states $\mathbb{M}$ (e.g. change in car's lateral velocity if it has longitudinal velocity, only possible to turn off bulb if its already ON).

Goal of the behaviour analysis in V&V is to explore and identify disparate states or operation modes of the system in terms of unique combinations of subsets of parameter value sets **I**, D_P and **O** w.r.t to a particular design solution's property value set i.e. a state $\mathbb{M}_{pr1}$ can be given in terms of a value set $\mathbb{M}_{pr1}$ = {x1 ⊆ D_X, e1 ⊆ D_E, u1 ⊆ D_U, Pin1 ⊆ D_P} w.r.t Po1 ⊆ **Po**. For verification to be successful, the set x1 should always be part of expected system outputs **X** otherwise the goal of validation is to identify new design solution such that the combination of Po1, e1 and u1 causing x1 to go out of set **X** can be constraint while respecting global property constraint on Po1.

Different scenarios can be created by defining system states w.r.t inputs through partitioning of input and output value sets of mapping **E** × **Po** × **U** -> **X** (combination of particular input and output values) e.g. a car's particular state can be defined based on its position, velocity and orientation which is induced by car being already in particular state (high velocity) and due to presence of certain features of external entities like humps on road changing input profile, high rate of turning at bend inducing centrifugal and gyroscopic forces on car.

Different solutions can then be sought for different scenarios such as implementing control of power distribution between a pair of output parameter(s), changing temporal characteristics of output power delivery rate, filtering particular disturbance etc. In other words, by analysing and comparing output parameter value obtained by use of existing design solution with respect to required parameter values, we can either apply a different type of design solution consisting new component configuration or make changes to property values of existing design solution.

3.7. Proposed Framework

As of present state of MBSE modelling tools, system model developers modelling system at high conceptual level, involving use of logical system entities, are unable to perform early verification of system by utilising the existing knowledge related to the problem domain unless supported by detailed domain specific simulations of prospective solutions realising the structure and behaviour of logical system entities.

Construction of high level architecture representing requirements can be informed by knowledge of which prospective design solutions can be used and therefore which physical constraints they can induce. Physical constraints can play vital role in determining whether system function can be achieved in real environment or not under different input conditions. e.g. when designing an automated door system it is essential to model the dependency between a pedestrian walking speed, sensor detection range and doors opening speed while considering the available door actuators performance limits. Also knowledge of potential solutions to the

requirements helps to identify loopholes in requirements which can then be amended until correct subset of solutions has been generated.

The knowledge of such constraints can also help to impose limits over the undesired behaviour or to identify limits of expected function's behaviours. System is usually analysed in a simulation environment, using the models of the design solutions (selected to realise its conceptual functions), to understand effects of physical constraints arising as a combination of internal and external factors. However, the simulation model of the system being designed may not be readily available because the design specifications required for instantiating properties of simulation model components are not readily available at the start of system model construction.

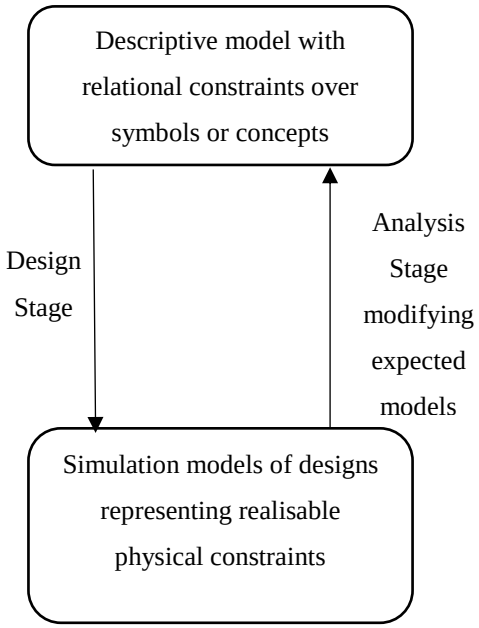


Figure 3-5 System model partitioned and linked by different modelling stages on bases of expected conceptualisation in descriptive models and actual realisable designs in form of simulation models

This research aims at developing a conceptual framework required to relate existing knowledge in descriptive models (e.g. models modelling function and behaviour requirements in relation to SAD) and domain specific designs[19] which can be used to perform following two primary knowledge based model analysis tasks,

1. Determine whether required functionality can be achieved by existing designs.

[19] Design refers to the available solution design in form of descriptive and simulation models

2. Identify limits and deficiencies of system's expected function behaviour based on various types of constraints.

These two tasks requires usage of knowledge representation and inference mechanisms. There are many ways of representing knowledge but in this research one approach from two main paradigms of AI have been selected namely use of ontology as well as first order logic along with SAT solver and machine learning especially using Neural Network Auto-encoders. The former approach requires manually modelling physical constraints of a particular domain (e.g. kinematics domain) whereas for the latter approach already existing physical or simulation models of the solution designs can be used. The main focus of this research is to adapt the model development methodology so that machine learning algorithms can be used where possible to aid the system developer.

Various other subtasks can emerge from the two main tasks e.g. the required design constraints (in terms of range restrictions on system property values) can be inferred by learning relation between domains of various requirement models (e.g. which are modelling requirements of a transportation vehicle for private or public domains) and the characteristics of physical properties and parameters of end product design. However this research utilizes the proposed framework shown in Figure 3-6 and Figure 3-7 to demonstrate the role of AI in achieving the first task only:

- Retrieval of descriptive design models by formalisation of concepts and relations between them required to encode the requirements, logical static, logical behaviour and physical architecture models at high level of abstraction as well as to make formalisation amenable to further domain specific specialisation for expressing low level physical constraints and for applying custom rules required for allowing only correct simulation model retrieval.
- Retrieval of designs as simulation models with same behaviour characteristics as of the partial behaviour encoded in the abstract descriptive design derived from the logical system architecture.
- Filtering retrieved designs based on the SAD (not implemented).

3.7.1. Representing and storing knowledge

The system developer constructs requirement and logical architecture model to represent required system functionality by instantiating ontology schema or meta-model discussed in section System Domain Meta-Model 4.1.1. and then uses those functions to build use case scenarios representing systems function interaction with external domain entities (environment,

user etc.) in functional flows[20] or logical behaviour architecture. The function decomposition of high level functions can then be done by using standard functions and flows e.g. function basis [65], this however is not implemented but can be easily done by specialising function and connector class from system meta-model. These low level functions are then used to retrieve descriptive models of physical subsystems.

Information related to generic (applicable everywhere for particular engineering domain e.g. generic types of gear) as well as domain specific (transmission types in automobile domain) usages of system's requirement and architecture model can be represented and stored as knowledge by instantiating ontology schema. Whereas known engineering designs can be stored based on their components behaviour data under different input scenarios by application of machine learning algorithm in form of Neural Network (NN) embeddings. However machine learning algorithm encodes knowledge in numerical format therefore relations between different designs (e.g. if one design is similar to other) can only be deduced by applying distance or similarity measures on numerical representation of each design.

3.7.2. Retrieval of physical descriptive subsystems

Modelling in first order logic (FOL) along with logic satisfiability solvers and constraint satisfaction algorithms (especially for mixed integer non-linear constraints) has been used [39] to construct physically feasible[21] physical subsystem architectures using rules to,

- Define functions based on port types (e.g. convert function should have port types pertaining to different engineering domains)
- Relate different type of functions to each other (e.g. Distribute function requires inputs from two other different functions)
- Relate functions to physical subsystems based on port information and
- Relate system's I/O values to mathematical relation between I/O parameter of physical subsystems for static scenarios (e.g. calculating net torque at end of lifting system based on electric motor's torque scaling by gears).

However in this thesis, definition of system meta-model (specifying relations between function, physical subsystem, port etc. in section 4.1.1.) is given in ontology which can be implemented by using either FOL or OWL – Ontology Web Language, but later allows for decidable

[20] The function flow model merges interaction information between external entities and system with decomposed functions to show interdependencies between functions when executing a particular use case.
[21] Physically feasible in the sense that they can be simulated under physical laws

reasoning[22] [66, p. 144] when used in restricted form as well as the instantiated models are more easier to understand when translated into a graphic form using RDF[23] (Resource Description Framework) graphs.

3.7.3. Retrieval and classification of simulation models

The retrieved physical subsystem architecture(s) can be in form of symbols and relation between these symbols with some of the main design property values defined (e.g. gear ratio defined but not gear damping etc.). The symbolic physical subsystems can then be allocated input to output behaviour from function behaviour defined in the logical architecture and then using clustering capability of neural network (especially of auto encoder to bring together similar information), designs having similar behaviour across two of its components (one mirroring physical subsystem's input port and other physical subsystem's output port) can be retrieved. The latter approach is implemented in section 5.4. The retrieved designs can then be integrated to form full system design. The retrieved full system design can then be inspected for violation of system design property or global property constraints by checking if they can be classified[24] together to SAD given in the system requirement model. This later inspection step is not implemented and is left for future work.

The knowledge based model analysis tasks at the centre of proposed framework ascertain if any physically feasible design exist to satisfy requirements and architecture model by acting as an interface and integrator for information flowing between descriptive model and predictions made by machine learning method. However for future work this instead can be used as data fusion process which can utilise either logical or machine learning framework or combination of both (with machine learning at low level providing probable predictions to logic) for particular subtask arising from the two main knowledge based model analysis tasks e.g. machine learning algorithm can be provide as input, the relations between descriptive models elements, modelling decomposition of particular function and its interaction with other functions using standardised flows [65], to rank retrieved designs (pertaining to same SAD). However this will require converting symbolic information to numerical information by using different adjacency matrix for each relation type and by using columns of a matrix to represent different attributes of a model element.

---

[22] With reasoners such as PAGOdA - http://www.cs.ox.ac.uk/isg/tools/PAGOdA/

[23] https://www.w3.org/TR/rdf-concepts/

[24] Multiple chained classifiers may be needed e.g. first classifier can be trained to classify subsystems with multiple types of property values (can be one or more than of same type) belonging to different SADs and then another classifier can be trained to give score for classifications belonging to each subsystem group forming one design

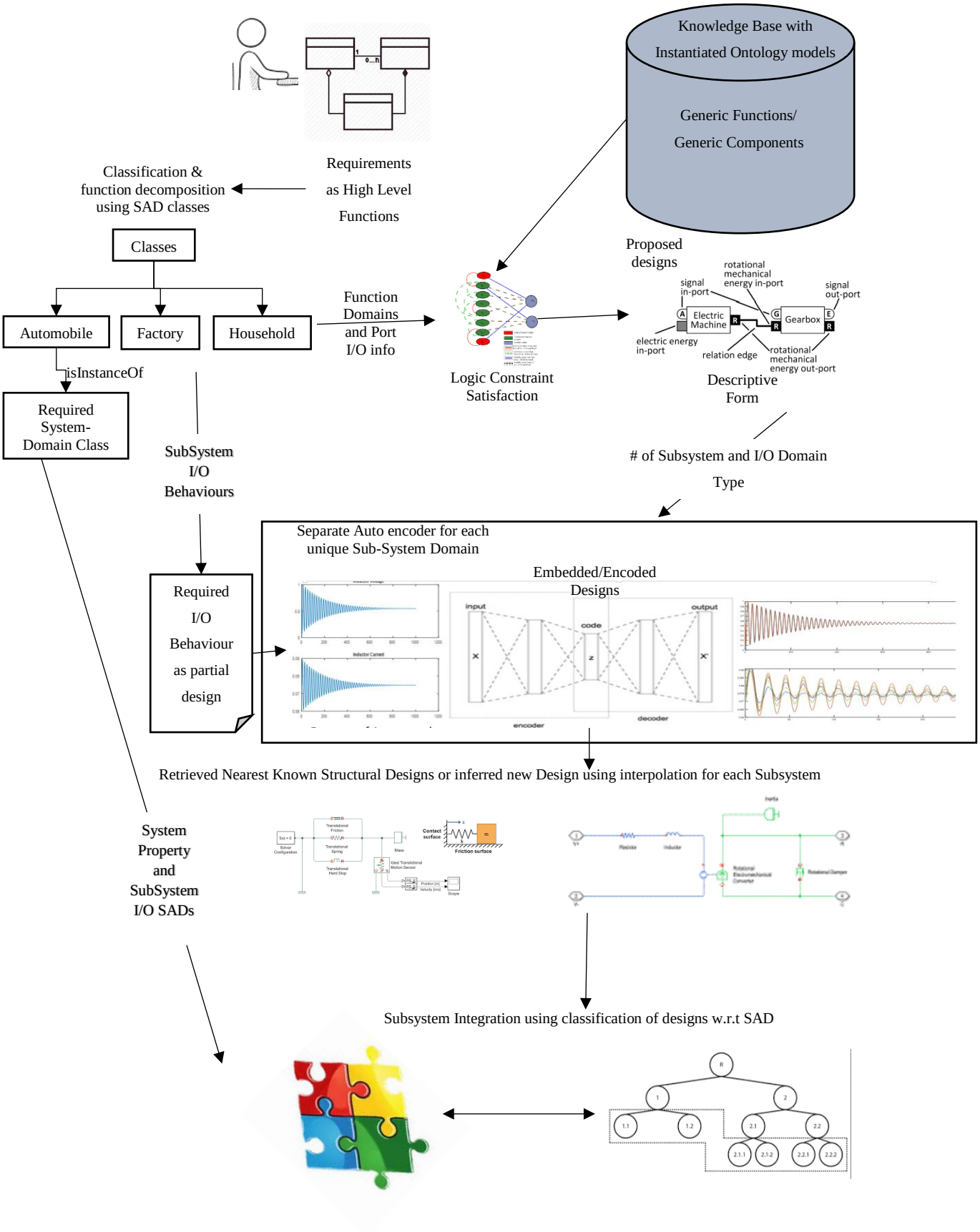


Figure 3-6 Illustration of relation between high level processes and elements of proposed framework using pictorial examples

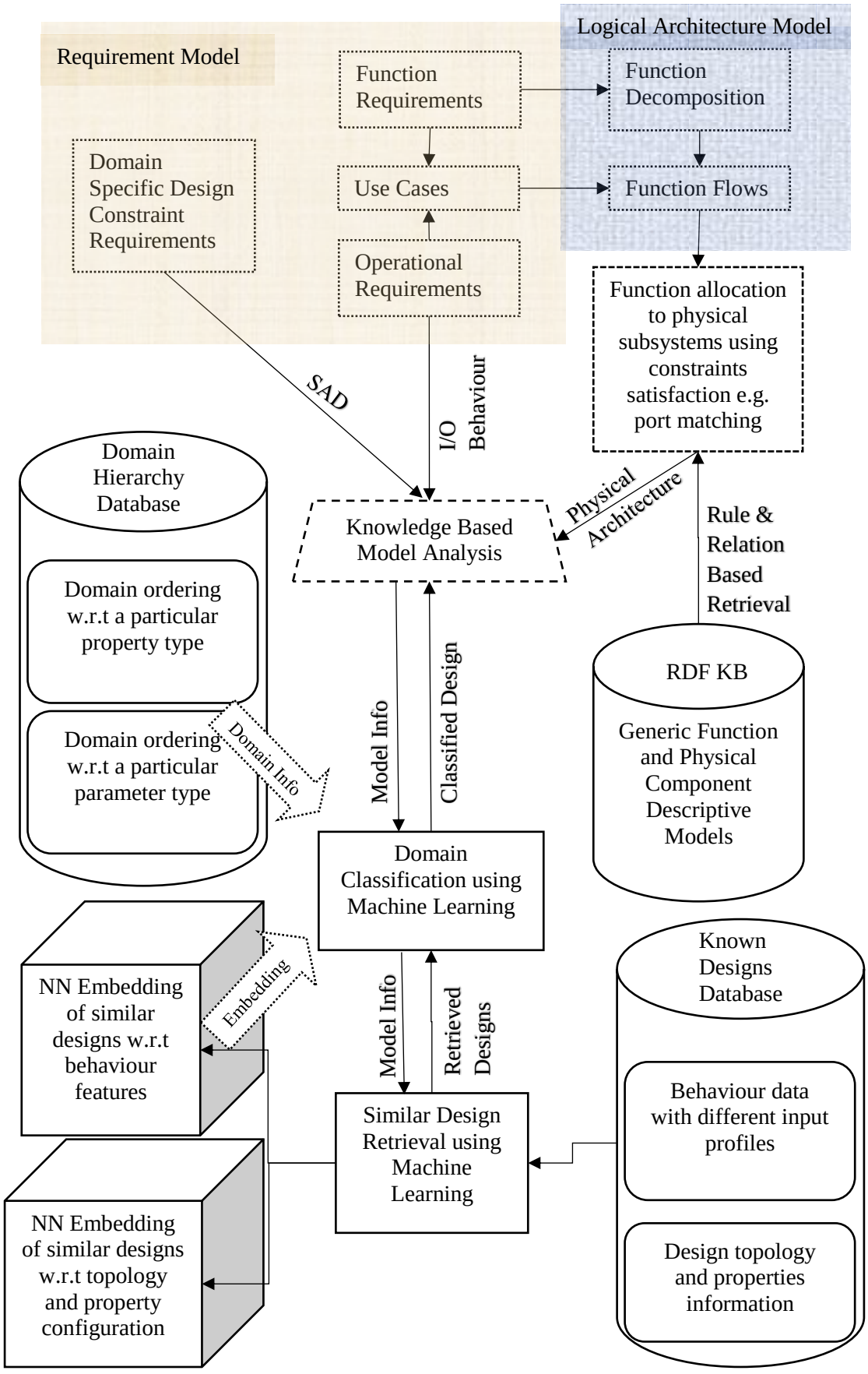


Figure 3-7 Proposed framework depicted using databases, process and their input-outputs.

Dotted outlined boxes represents manual model development, small size dashes line boxes represents deduction using logic constraints and long dashes represents use of both machine learning algorithms and logic or constraints satisfaction solvers. The solid line boxes represents inference by machine learning algorithms.

The framework shown depicts neural network training data originating from database (shown in cylindrical shape in Figure 3-7) or existing knowledge whereas the information from manually constructed model becomes test dataset for machine learning algorithm. In the experiments of section 5.3. only one particular input profile is used in generating data but with different type of input sources and methodological approach to separate Neural Network (NN) embeddings into different categories has not been established and is left for future work.

3.8. System Application Domain (SAD)s and their Hierarchy

So far developers have only been able to benefit from manually defined meta model and design rules e.g. which function type decomposable to which other type and which function can connect to which. However they have been unable to analyse logical model as per abstract ground truth without referring to simulation models derived using physical architecture. By noticing that high level functions are decomposable into low level functions and it is possible to decompose high level value domains (e.g. value limits) over parameters of low level functions and over properties of entities executing those function, which implies that due to this intra connectivity high level functions parameter refers to ground truth[25] indirectly. Hence by using this decomposition relation between high level parameter characteristics and low level property & parameter characteristics (encompassed by existing designs) we can check if existing working designs are entailed by high level logical model. SAD is based on this insight of relating characteristics, such as value domains (by use of machine learning or by other means), of parameters and properties residing at different level of system model abstraction.

[25] Admissible value domains of high level function parameters w.r.t ground truth

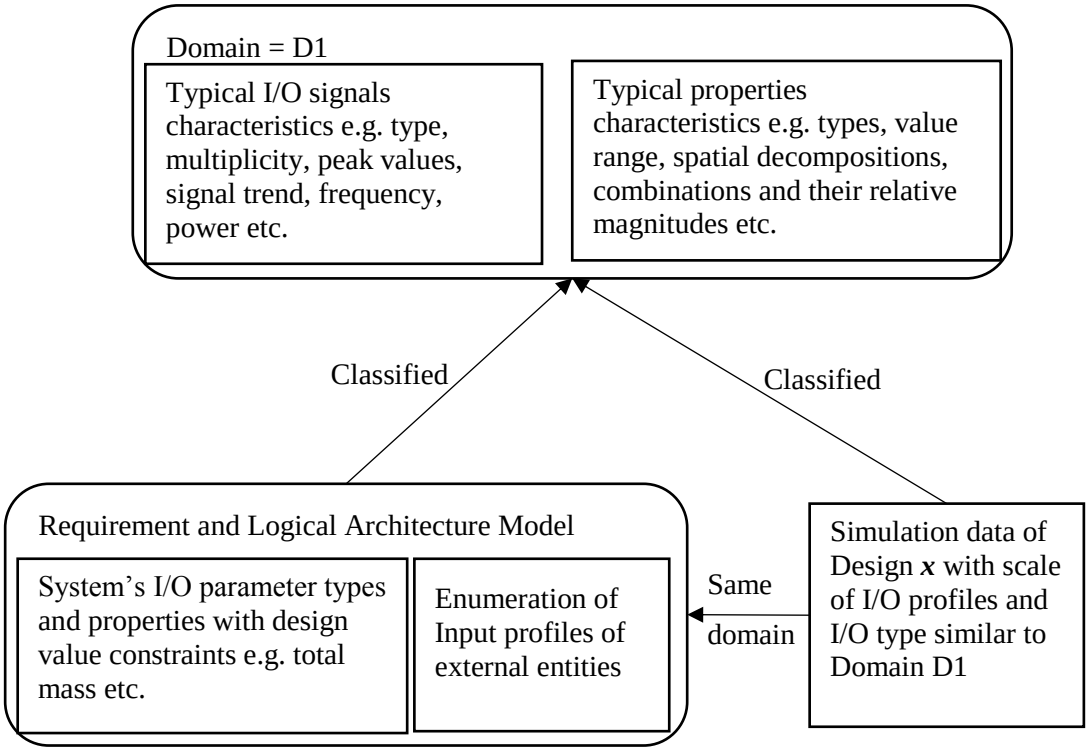


Figure 3-8 Design solution represented by simulation data and high level descriptive models related to each other by using classification to a particular SAD

A SAD helps to classify a system to a known group of designs by determining if that design's common characteristics are same as of known design both at descriptive as well as behaviour level. Therefore in essence SAD is defined by known designs and their common characteristics. But by relating different SADs w.r.t particular property and/or parameter characteristics, a selection criteria can be developed for comparing models of known designs. As shown in Figure 3-8, for any system with known typical combination of characteristics I/O parameters and system level properties, a domain can be assigned which encapsulate those typical characteristics. Simulation data belonging to a model of a particular design and an instantiated descriptive system model (or its elements like function, sub-function etc.) can belong to same domain if their parameter and property characteristics also belong to the same domain. Domains are not disjoint and can therefore overlap with each other if some of the characteristics of property and parameter are same. These domains can be relatively arranged in an order based on one or more than one parameter and property characteristics of existing systems used to define those domains (as shown in Table 3-1). This ordering can then be used to order existing designs (or simulation models) as well as descriptive model elements which helps in their selection during construction of a new system model by determining their domain similarity with system domain. Domains can also be used to restrict value domains of parameters and

properties of the descriptive requirement and logical architecture system model which has referenced that particular domain.

Max value ordering criteria	Household	Car	Truck
Domain property = Inertia	Low	Medium	High
Domain parameter = Velocity	Small	High	Medium
Domain parameter = Force	Small	Medium	High
Domain parameter = Electric power	High	Low	Medium

Table 3-1 Relative ordering between three domain based on only property and parameter's value ranges

If the domains are further specialised w.r.t entity types (e.g. Household can be specialised into Household: Adult, Household: Child), then in those domains existing designs and model elements can also be arranged into different levels of abstraction (e.g. Force parameter value range for dial or switch HMI system will be different for adult or child with child domain system being more *specialised*). Domains can also be specialised based on system types i.e. systems with same input but different output types (e.g. Household: Lift and Household: Lighting system both are likely to have electrical input but with different power characteristics). These specialisations can help to decompose descriptive model elements more consistently using rules such as all model elements at same level of abstraction should belong to same domain, domain of all physical subsystems or their corresponding functions can only be specialisation of system domain. Such system type based specialisation can also help to complete partial or missing information in requirement models (e.g. required velocity in mass lifting system may be unknown but can be inferred from same type of systems in same domain).

Chapter 4 Aiding System Model Development using Logic

A system model is a composition of function, structure and behaviour models. Assisting the designer to retrieve model elements to either complete a partial system model being designed or to replace and correct some part of it, requires using relations between different types of models and model elements constituting system model. Descriptive or logic based knowledge representation formalise function and structure models for application of different types of reasoning (e.g. using sub-class, sub-property, transitive etc.) as these models are mainly constructed by relating concepts with relations as well as by relating concept attributes to their respective concepts. The reasoning is required to retrieve existing knowledge (in form existing system models) related to same domain as system being developed.

A general model of system model development process can be constructed based on the knowledge transfer (of number, type and value domain of parameters and properties[26] and relations between them) required from "as-is" system-domain model to the "to-be" system-domain model as shown in Figure 4-1. Distinction between SAD and 'as-is' system-domain model stem from their usage i.e. former is used to categorise the later by leveraging characteristics associated to attributes of both system and domain.

However, in this approach we have assumed that particular class of system solution is available captured in form of 'as-is' [67] system model for a particular type of 'as-is' domain model (which captures environment configuration for a range of system usages or applications). The 'as-is' system-domain model represents relation between the nominal environment and input-output ports of standard sub-system(s) of system in that domain e.g. car's nominal environment is road and had ABS function which interfaces with road. This model captures high level relations between external system ports and the nominal environment in which the system operates. Specification of the ports in 'as-is' model is given in the physical architecture which can consist of its domain type, position, orientation (interface side), parameter type, further parameter attributes or features (e.g. direction, component resolution, frequency etc.) etc.

System requirements are elaborated user concerns which are reflected by either adding entities (external user, environment features etc.) and use case scenarios to the as-is domain model or by adding additional functions to the as-is system model. This extended form of 'as-is' domain and system is thus called the 'to-be' domain and system model.

[26] Difference between property and parameter arise from the fact that former is assumed to be constant w.r.t space and time in ideal conditions i.e. no effect of external phenomenons like force, temperature etc. In reality however property can change but at very slower rate compared to parameter unless it is design intent (e.g. change in mass of fuel rocket).

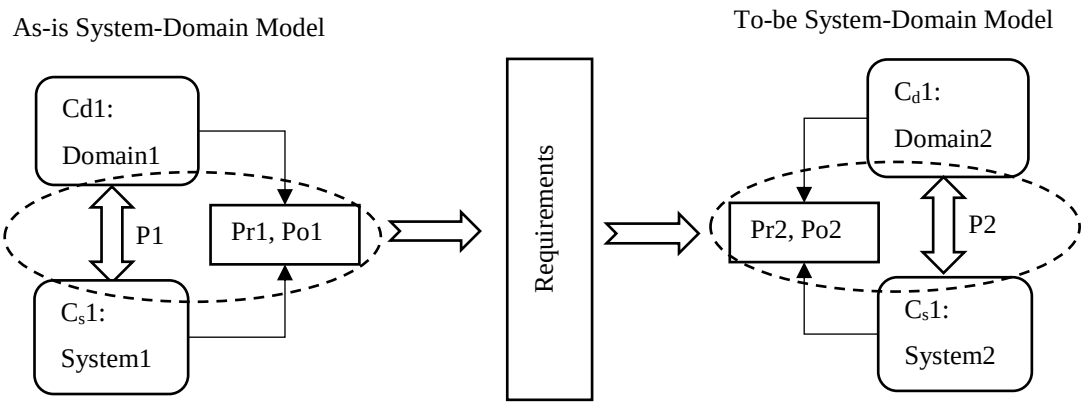


Figure 4-1 Illustration of knowledge transfer of model elements from the "As-is" to "To-be" system-domain model.

In Figure 4-1, C_d1 is a class from set of domain classes and C_s1 is a class from set of system classes. The relations as well as properties & parameter information of the “as-is” system-domain model are transferred to the “to-be” system-domain model based on the new requirements.

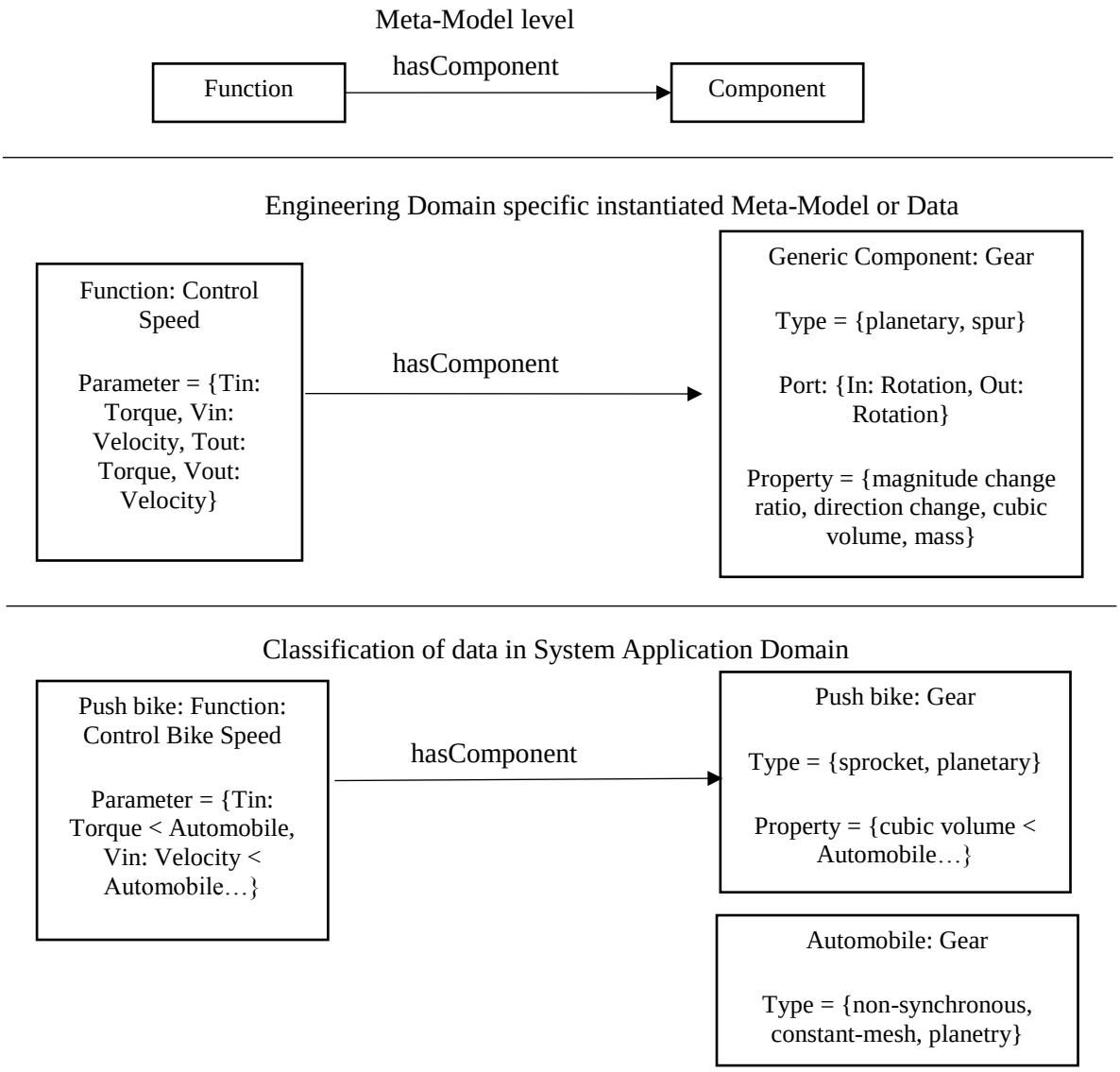


Figure 4-2 Specialisation of function and component using Engineering Domain and SAD.

Storage and retrieval of existing models related to "as-is" system-domain as well as generic logical and physical architecture models requires storing information related to types of functions, types of components, types of parameters, types of properties and relations between different types of elements (function to function, function to component etc.). Each model element is also required to be classified under SAD, meta-model element type (function, component) and generic engineering domain types (a function can be of type "Convert" or types from functional basis [65]) as shown in Figure 4-2. Each of such knowledge element can serve one or more purposes e.g. the function decomposition in static logic architecture requires use of pre-defined functions (in terms of number and type of parameter etc.) as well as relations between functions (abstract to detail, classification of functions to known domains etc.). Similarly, the physical architecture model development requires use of stored solutions for particular system type (mechanical, electric etc.) and SAD (e.g. selection of components with

properties and parameters under same SAD as shown in Figure 4-2). Above all, to relate each low level function (with a defined behaviour) to a particular generic component or subsystem requires giving unique identity to those functions. Such identity is formulated based on the combination of categorisations of the function to different classes or domains (e.g. a function in logical architecture can be put through domain specialisation sequence shown in Figure 4-3) and relation to other functions in the same models.

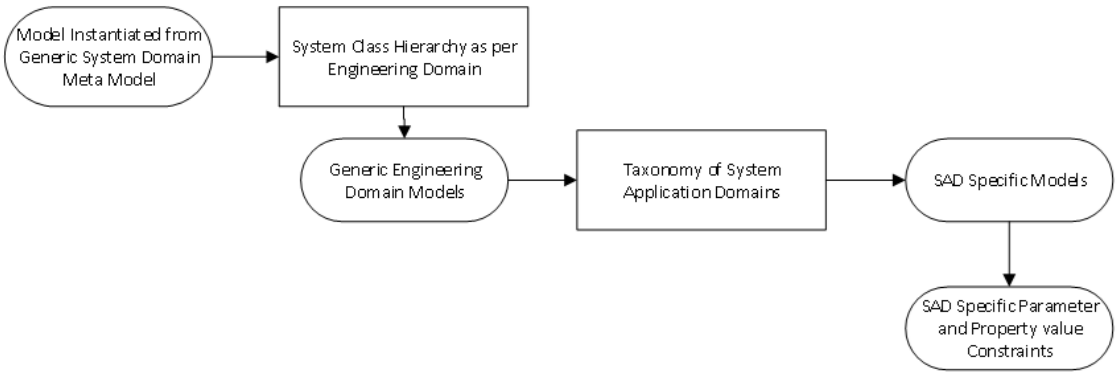


Figure 4-3 Specialisation of a model or its elements by application of different domain types (represented in boxes).

The proposed framework in section 3.7. shows utilisation of known system models. However, the representation in which the model knowledge is stored needs to conform to a common schema or template against which rules for retrieving the instantiated models can be written. The trade-off between expressiveness of the chosen knowledge representation formalism (OWL, RDF(s), first order logic etc.) and extent of guarantee of completeness and soundness of inference[27] applicable on the selected representation.

The meta-models or schemas shown in following sections can be tailored and refined for specific application domain based on tacit or experiential knowledge of the engineers that they have gathered by performing same task multiple times. It has been claimed in [68, pp. 455-456] that 80% of design engineering activities are repetitive and routine e.g. in car design structural decomposition is well known. Whereas the domain specific knowledge related to the system's behaviour can only be found in domains with strong domain theory [31], where it is possible to model and analyse system behaviour in all input/output scenarios with required detail, whereas domains with weak domain theories require use of probabilistic methods to estimate property values as shown with the use of NETSYN [59] for generating configuration of personalised computer as per required performance. However in both the cases the knowledge related to high

[27] when reasoning or inferring new and correct relationships between knowledge elements based on given relationships and classification of knowledge elements

level summarisation of system function and structure (without considering property values for weak domains) in form of logical and architecture model can be represented and stored.

4.1.1. System Domain Meta-Model

Ontologies provide a shared vocabulary to system model developers by providing specification of concepts and relations (between those concepts) in form of meta-model or schema which ensures consistent communication between them. It also enables the reasoners to reference the schema when reasoning over supplied facts (i.e. in form of instantiated schema) to infer new facts as well as when organising those facts into a knowledge base. The pre-stored libraries of models can also be organised and reused by referencing the ontology specifying relations between those models and with their usage context (modelled as requirements ontology), as done in [26].

Toolkits like Jena[28] enables construction of OWL/RDF(s) ontologies and usage of reasoners for deriving additional facts from the instantiated ontology as well as application of custom rules and query languages like SPARQL[29] for querying over the stored knowledge. Jena stores ontologies in triple pattern (subject-predicate-object) in a relational database. Reasoning can therefore be performed over the stored knowledge by combination of ontology based inference using the axioms specified over the ontology, e.g. subsumption reasoning, transitive reasoning, and by rule based inference through the use of SPARQL query patterns.

Schemas for system domain ontologies, as shown in

Figure *4-4*, is created from the partial constructs of SysML's Block Definition Diagram (BDD), Internal Block Diagram (IBD) and Activity Diagram. These schemas can be instantiated to store generic and domain specific function and structure knowledge. Only partial constructs are used here to enable efficient reasoning and also to capture sufficient information or conditions required for writing rules linking different type of model elements. Discrete or logical behavioural aspects related to software design modelling, such as control flow in activity diagram, will be considered by further elaboration of meta-model by considering the corresponding SysML constructs in future.

[28] https://jena.apache.org/index.html
[29] https://www.w3.org/TR/rdf-sparql-query/

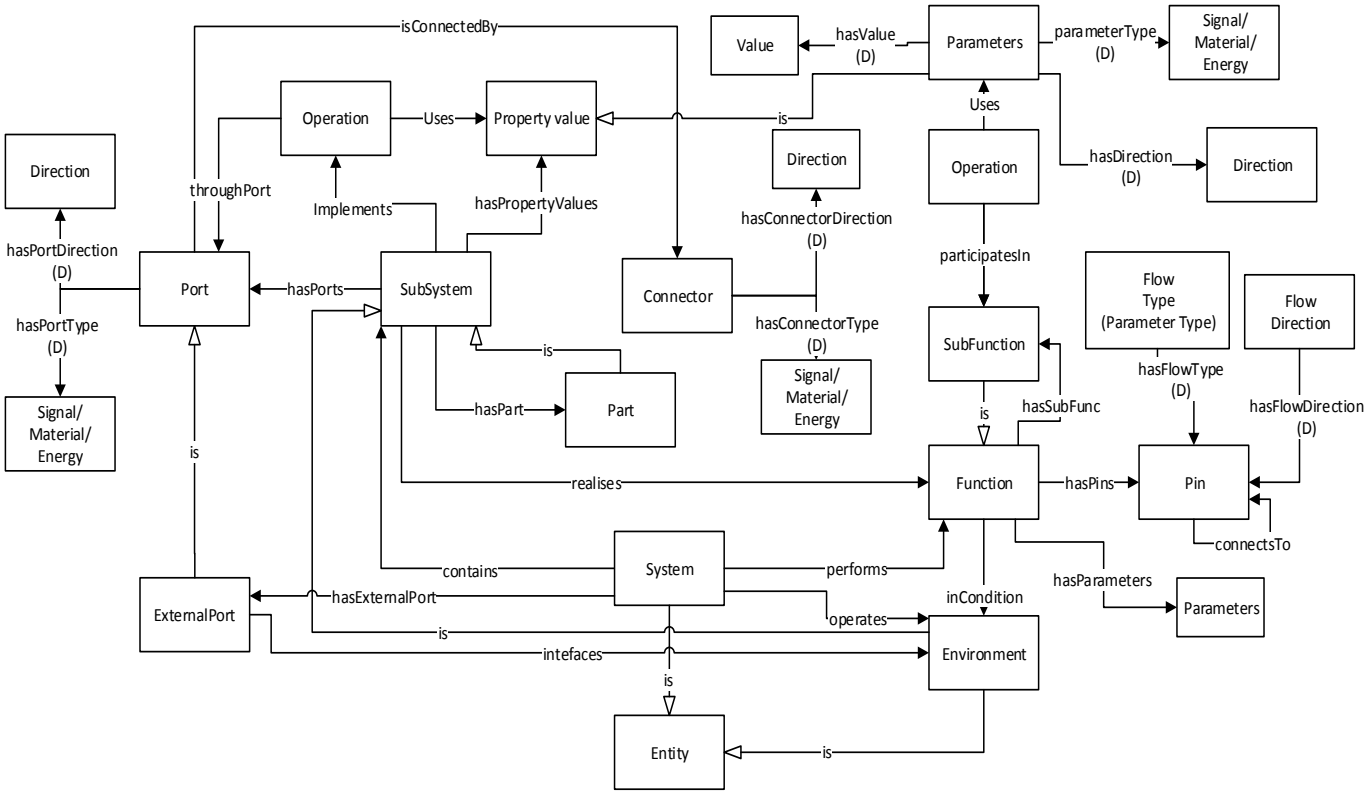


Figure 4-4 Ontology schema for system domain meta-model

In this ontology schema square or blocks are the classes, the lines or edges with the filled arrow head represent object properties, the line with unfilled arrow head represent generalisation or subclass relation and black arrow lines with suffix (D) in their annotation are the data properties. This schema can be instantiated by creating individuals of particular classes and using the relations between those classes. Further constraints on the relations, such as relation type (symmetric, transitive, functional etc.) and use of cardinality restrictions (has at least one value, has all the values etc.) for the object of the relation can be also be specified where necessary for making queries or rules more flexible.

The property values of sub-systems (and its parts) represent physical attributes as well as other variable and constant attributes. The parameters of the operations are variables (i.e. subset of property values) which are exchanged between the subsystems performing a particular function. Operations also helps to bind subsystem to several sub-functions by pairing the property values with parameters. Operations are the set of characteristic functions of the subsystem determined by the joint active state of all of its parts based on the external input conditions. In other words, operations are set of enumerations of parameter values for each property value and external inputs combination. The connectors provide physical medium on which parameters flow, hence

type of parameters helps to decide type of connectors and ports. Here port is considered as part of the whole structure, also called 'Proxy ports', rather than as 'Full ports' which can be parts of the subsystem and can have its own behaviour. The port's function is restricted to only interface material, signal and energy flow of the subsystem or its components to the other subsystems and components. Direction attribute of ports, connectors and pins can have value of in, out and in-out. This ontology may need to be corrected when implementing it to ensure correct inference of relations e.g. to store "operation" in RDF triples (operation is defined by logic or mathematical condition over time and/or space e.g. Distribute = {∀ F1 ∈ Functions, disj s1, s2, s3 ∈ Space, disj x1, x2, x3 ∈ F1.hasParameter, x1.hasDirection = In, x2.hasDirection = Out, x3.hasDirection = Out: x1(s1) = x2(s2) + x3(s3)}), use of rules will require which can generate different triples for each numerical value belonging to the parameter value set. Also this type of behaviour encoding (in set form) can be done using higher order logic such as in Alloy Analyser in implementation of higher order relations of semi-qualitative kinematic framework (in section4.2.2.).

4.1.2. Formalisation of Textual Requirements

Intelligent computational support for requirement definition and analysis can be provided by using a formalised requirement model. Such model can support model validation reasoning such as checking consistency between requirements and the interactions (e.g. between environment entities and system components) addressing the requirements, between requirements and component properties as well as with testing results validating the performance and operational requirements.

The model in form of a schema, shown in Figure 4-5, provides a consistent input for all other knowledge based model analysis tasks, especially:

- Matching with existing "as-is" system-domain model and then identifying relation between given and known entities as well as between their parameters and properties.
- Retrieving generic structure and function models by matching their engineering domain type (with those given in requirement) or by using superclass relations of the required engineering domain and taxonomy of engineering domains (which can also be modelled using ontology).

Requirements can have performance and design constraints on function's parameters and structure's properties respectively (specified as numerical constraints). The former specifies constraints on functional parameters under given operational or environment conditions. The latter is used to filter the solutions derived based only on their functional feasibility. Other requirements can also be defined in terms of interface and interaction medium such as type of

inputs required on control panel, mechanical interface for linking components in assembly, type of fuel required, format of information transfer etc. These requirements can be included as subtypes of perform, contain and operate requirements.

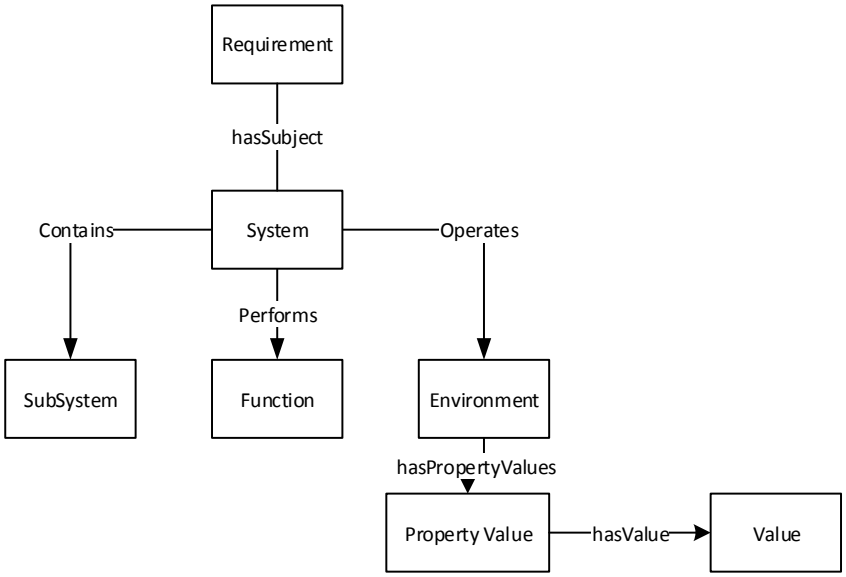


Figure 4-5 Requirement ontology schema

The operational requirements can be represented in transfer function form [56] i.e. A requirement has input and output entities such as system's external interface, interaction medium for interacting with an external system, external physical system etc., with requirement type stated as relation between them, which can be perform, interface, operate etc. These relations can be characterised by quantitative or qualitative attributes. Example of such requirement can be

"The Car shall operate using Diesel Fuel with [fuel efficiency] of not less than 10 Km/litre over the Asphalt road."

in which subject system is Car, input entity is Diesel Fuel, output entity is Asphalt road, relationship is fuel efficiency and its attribute is >10 Km/litre.

This requirement representation can be converted into a formalised form using ontology as shown in Figure 4-5 which specifies different types of requirements as relations.

System *perform* vibration damping – Functional requirement

System *operate* water [x metre] (depth) – Specified Operation requirement

System *contains* emergency indicators – Structural requirement

Emergency indicators interface with humans – Interface requirement as sub-property of structural requirement i.e. interface subPropertyOf contain.

Different predicates (perform, operate etc.) indicates types of requirements which can be used to infer lower level of functions and subsystems or components. Operational requirements can have property value(s) (given in curly bracket) with a value (given in square brackets) which can be used to infer subsystem properties. Case based knowledge defined in terms of existing models (instantiated from meta-models) of requirement, structural and functional models can be related to the given problem in form of new requirements in a similar way to the use of recursive case based reasoning of [20] (described in section 2.3.2.) and use of design patterns for model library organisation as in [28] (described in section 2.2.2.).

4.2. Logical to Physical Architecture Elaboration

The relations between the required model elements (given in requirements) and elements related to engineering domain classes as well as SAD classes, above and below in the hierarchy, can be used to retrieve high level structural and logical models e.g. Car has superclass relation with automobile therefore will also have transmission subsystem between power source and external output port.

The information from generic high level models, pertaining to the requirement's engineering domain, can be used to determine all the minimally required functions whereas the matching domain specific models in the SAD can be used to cater for special structure and function features required by user and necessary for operational environment under the SAD e.g. class relations in SAD can be used to retrieve specific model elements e.g. Car has subclass relation to rally car which can be used to elaborate required rally suspension sub-system (if required by user) with properties from rally car model. This retrieval can be done by using class hierarchy relations, property hierarchy relations, domain specific rules matching port types and specifying canonical connectivity structure between particular component types (as in Figure 4-6).

Based on the retrieved and given functions and subsystems further functional and structural decomposition can be performed to identify sub-systems, sub-functions and operations provided by the subsystem for realising the functions. This decomposition or elaboration can be performed by using rules specifying component order to achieve required transformation between internal and external system parameters by using operation provided by functions (e.g. value magnitude change such as torque scaling with gear ratio, energy domain type change such as electrical rotational to mechanical linear, relative structural orientation change between input and output port etc.). The decomposition needs to be performed so that the lowest level function can be represented by generic components (e.g. gears) and generic connections (e.g. pin slot

joint). Based on the decomposed sub-systems and sub-functions, component properties as well as operation parameters can be inferred which can then be used to identify the pin and port types as well as the functional flow and connection types.

Relations between sub-functions, i.e. flows, can be inferred to create functional flow e.g. Adaptive Suspension controller –"hasFunction"-> Adjust damping ratio –"hasPin"-> data pin –"connectsTo" -> data pin <-"hasPin- Elevation measurement function <- "hasFunction"- Data collection and measurement unit. The usage of decomposed functions or operations in different functional flows (scenarios) by their respective subsystems or components determines the type of input and output parameter(s) exchanged between them as well as the interface required for those exchanges (Power transmission controller performing elevation measurement function takes tilt signal from inertial measurement unit sensor connected to car body therefore if controller is also performing traction control function for turning scenario then it will take angular precession signal from same sensor acting as a interface port). Therefore, connections and interfaces between sub-systems (or its parts) can be inferred based on parameter types exchanged in a particular flow between functions (or its respective sub-functions and operations) realised by those sub-systems (or its constituent parts).

4.2.1. Retrieving Generic Physical Components by Specialising Meta-Model

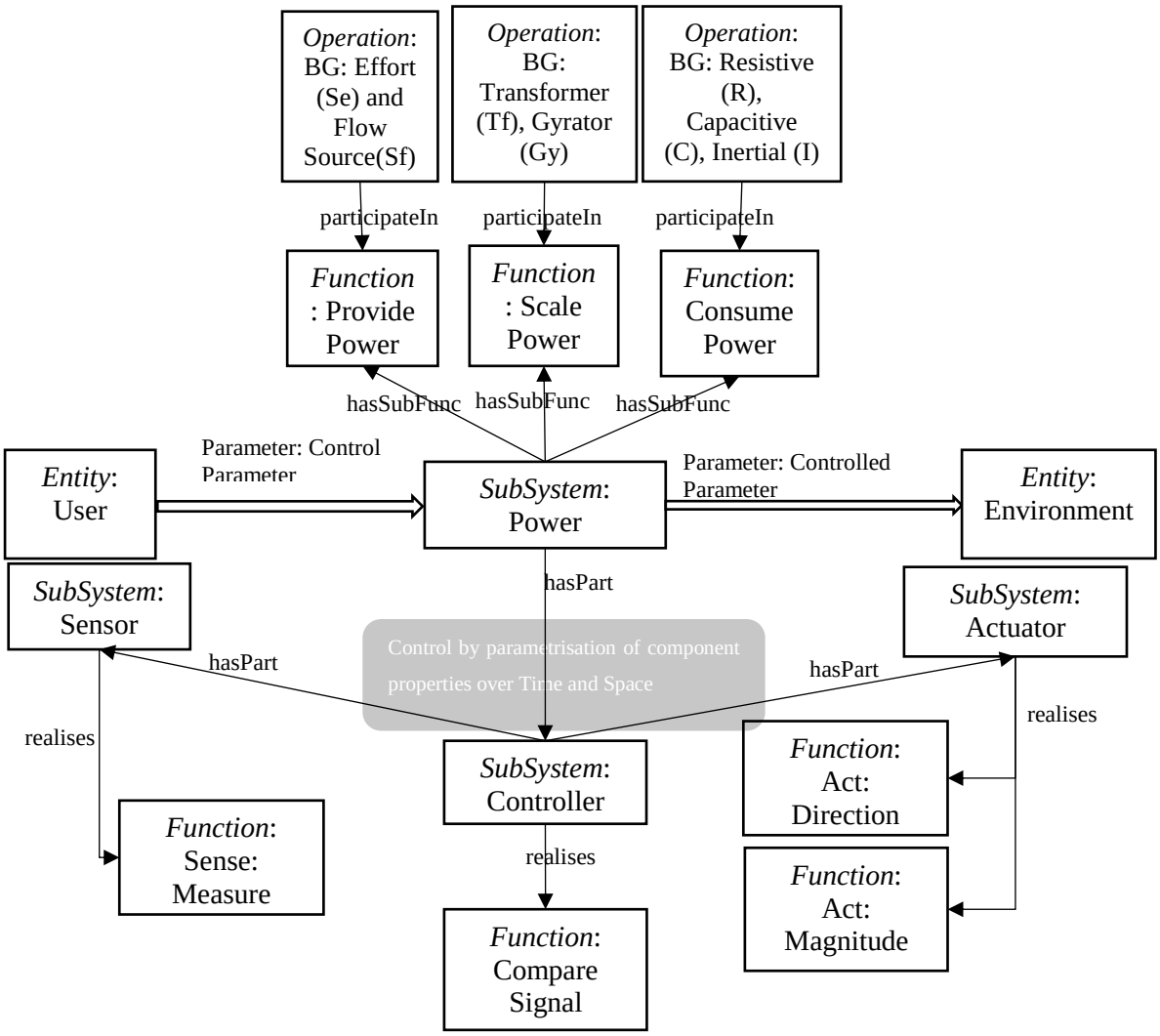


Figure 4-6 Specialisation of generic System-Domain meta-model using Mechatronics and Bond Graph concepts

This meta-model specialises the system-domain meta-model given earlier into Mechatronics related components and functions. Every element in this meta-model is either a sub-system or a function and thus has the same type of elements (and relation between them) as defined for a function or sub-system in system domain meta-model e.g. system has external (I/O) port or interface, which is realised by a type of generic sub-system, and this interface act as gateway for input and output power transfer between system and environment. Based on above meta-model many rules can be written based on specialised or sub-type of each subsystem, function and their relations so as to map those elements to energy paradigm conceptualised in Bond Graph (BG) methodology (bond graph is a dynamic system modelling methodology developed by Paynter [69]).

Actuator is a specialised type of internal subsystems which influences main energy transformation process's magnitude (i.e. balance of effort and flow variable) and/or direction (unidirectional, bidirectional, blocking or kinematic directions like clockwise etc.) and have an interface with a controller (algorithmic or manual). Sensor interface on one side with controller as well as with other internal and external subsystems (e.g. sensor for measuring internal process temperatures and for monitoring outside environment). On the other hand, both components of this meta-model must have defined domain, dimension and directionality of the parameter being exchanged. All the elements of the instantiated model should satisfy the 'as-is' system-domain constraints (e.g. if 3d motion is considered as system output then control or limits should be modelled on all 3 dimensions as well as sensing of the actively controlled motion parameters should be modelled as well).

The nominal function flows or sequences from behavioural logical architecture corresponds to use of functions forming a flow of energy with logical behaviour constraints or operations imposed by magnitude control and direction control functions.

Power transformation subsystem represent major components realising the function(s) which either interface with control input or (in)directly affects the controlled parameter (i.e. parameter of system common with external entity). As demonstrated in example in section 3.1. , functions such as Change amplitude can be classified under "Act" type which indirectly affects the controlled parameter Lm1, Produce function along with Change amplitude function scales the power from source to affect the output hence together can be classified as "Scale Power" and the Compare function interfaces with control parameter Inp1 therefore was given type "Sense". Each of the functions realised by power transformation subsystem can be represented as an operation in form of BG element as (shown in Figure 4-6 for 3 main functions) especially in the context of design of mechatronic where component behaviour is continuous or discrete between two states (e.g. switch). By only using non-energy storing BG elements such as Se, Sf, Tf, Gy and R and by following BG construction methodology as given in [70], it is easy to write simple rules or constraints to deduce a physically valid parameter mapping between control input/power source to controlled output and each of those BG element operations can then be realised by generic physical components (e.g. gears, combustion engine, DC motor etc).

A BG methodology based framework of constraints required for automated function flow[30] construction using operations corresponding to BG element is given below,

[30] Represented by linked chain of operations

BG Element ∈ Op, we are assuming that operations are represented by BG elements from here on, but definition of operation can be extending by applying domain specific concepts e.g. for mechanical domain outputs, orientation change between input and output ports applied by different type of gears.

Objective: Construct an ordered list (Ls) of operations which satisfies energy conservation constraints and whose first and last elements are: first(Ls) = Pso && last(Ls) = Psi respectively, whereas it uses 2-port Bond Graph elements (BG_Element), Transformer(Tf) and Gyrator(Gy), as other intermediary components.

Psi, Pso, Gy, Tf ∈ BG Element, where Psi = Power Sink, Pso = Power Source, Gy = Gyrator and Tf= Transformer

Psi or Power Sink is only used to represent the output of last BG element in the list and to verify energy conservation constraint and in no way represents energy consumption or dissipation by resistance, friction, damping etc.

InDomain (BG_Element) ∈ Domain and OutDomain (BG_Element) ∈ Domain

InVar (BG_Element) ∈ {Effort, Flow} and OutVar (BG_Element) ∈ {Effort, Flow}

Where Domain is union of electrical and mechanical domain variables i.e.

{Electrical ∪ Mechanical} = Domain,

Domain set can be extended with other domains such as hydraulic, thermal as well as specialisation of Mechanical domain as rotational and linear.

Voltage ∈ {Effort ∩ Electrical}, Current ∈ {Flow ∩ Electrical}

Force ∈ {Effort ∩ Mechanical}, Velocity ∈ {Flow ∩ Mechanical}

Elements can be therefore connected to link power source to output element by matching their input and output domains.

Effort source can be used for selecting value for output flow for a given value of output effort and the input power of the system whereas Flow source can be used in opposite manner.

OutEffort(OutVar (Fso)) ∈ {PowerIn(Sys) / OutFlow(OutVar (Fso))} and OutFlow(OutVar (Fso)) = {Admissible range}

OutFlow(OutVar (Eso)) ∈ {PowerIn(Sys) / OutEffort(OutVar (Eso))}and OutEffort(OutVar (Eso)) = {Admissible range}

where {Eso, Fso} ≡ Pso where Fso = Flow Source and Eso = Effort Source/Sink

PowerIn(Sys), In/OutFlow (Flow), In/OutEffort (Effort) ∈ {0…N} for some finite N ∈ ℝ and Sys is singleton instance of System.

InFlow (InVar(Psi)) ∈ {0…N} and OutFlow (InVar(Pso)) ∈ {0…N}

InEffort (InVar(Psi)) ∈ {0…N} and OutEffort (InVar(Pso)) ∈ {0…N}

InEffort and OutEffort excludes OutVar(BG_Element) and InVar(BG_Element) respectively from their domains as well as InVar() and OutVar() relations should excludes Pso and Psi respectively from their domains.

Following energy conservation constraints can be applied over the system input and output as well as over the I/O of each intermediately element

PowerIn(Sys) ≡ OutEffort(OutVar (Pso)) X OutFlow(OutVar (Pso)) = PowerOut(Sys) ≡ InEffort(InVar (Psi)) X InFlow(InVar (Psi)), assuming 100% efficiency

For Intermediate elements:-

OutEffort(OutVar (BG_Element)) X OutFlow(OutVar (BG_Element)) = InEffort(InVar (BG_Element)) X InFlow(InVar (BG_Element)), assuming 100% efficiency

Every Bond Graph Element has two inputs and output ports containing Effort and Flow variables. The specific types of Effort and Flow variables at the input and output port along with the relation between the input and output variables defines the specific type of Bond Graph Element e.g. a gear is a fTf because it proportionally up-scales output flow (angular velocity) w.r.t input flow (angular velocity) and downscales output effort (Torque) w.r.t input effort (Torque) based on gear ratio (scaling parameter) whereas eTf has opposite effect on output variables w.r.t input variables. Tf and Gy are separated into two further specialised types each i.e. eTf, fTf ∈ Tf and eGy, fGy ∈ Gy. Bidirectional causality concept of Bond Graph methodology can be applied in definition of these specialised types of Tf and Gy e.g. by noticing eTf has opposite scaling between Effort/Flow variables w.r.t fTf and when input variables of a BG element are considered as output the eTf becomes fTf and vice versa i.e. when type of scaling on the variables types is reversed.

If BG Element = fTF => OutFlow(fTf) = m × InFlow(fTf), OutEffort(fTf) = (InEffort(fTf) × InFlow(fTf))/ OutFlow(fTf)

If BG Element = eTf => OutEffort(eTf) = n × InEffort(eTf), OutFlow(eTf) = (InEffort(eTf) × InFlow(eTf))/ OutEffort (eTf)

If BG Element = fGy => OutFlow(fGy) = m × InEffort(fGy), OutEffort(fGy) = (InEffort(fGy) × InFlow(fGy))/ OutFlow(fGy)

If BG Element = eGy => OutEffort(eGy) = n × InFlow(eGy), OutFlow(eGy) = (InEffort(eGy) × InFlow(eGy))/ OutEffort (eGy)

where multipliers m and n are constant with $\{m, n \in \mathbb{R} | m, n>0\}$

m and n are chosen by satisfying InFlow X InEffort = OutFlow X OutEffort and m X n = 1

In other words, the output flow of fTf is up scaled version of the input flow and as an affect the output effort is scaled down.

In case of representing the Change amplitude function (from section 3.1.), where a signal is controlling the change in electrical power, modular transformer and gyrator BG elements can be used whose scaling factor is varied by an external signal.

Power source, Transformer and Gyrator elements can then be mapped to generic components. Components are pre-defined by assigning engineering domain types to their input and output parameters. These components can then be chosen based on matching input and output domain to domains of each of BG element in the generated function flow e.g. an operation with type transformer with input and output rotational mechanical domain can be substituted by a gear with gear ratio corresponding to multiplier of transformer. Different type of gears e.g. worm gears, can be defined by assigning spatial orientations to parameters of operations and then defining spatial relations between the orientations imposed by operations e.g. if we can assign positive x and y directions to parameters of a mechanical operation then a worm gear will apply 90 degree rotation to input parameter i.e. if input parameter is along x direction then it will be applied counter clockwise rotation.

To discriminate and choose between two very similar components based on their dynamic property values w.r.t system's control and controlled parameter features (e.g. if input is varying like sine wave only then dynamic properties will have an impact), abstract dynamic states of the system can be modelled. These components will need to be defined by inclusion using inertial, resistive and capacitive properties (modelled as qualitative relation called *influence*[31] between the variables and their derivatives [71]) and abstracting their value trends w.r.t time (e.g. linear, rising exponential to constant, dying exponent to constant etc.) using rule of thumb knowledge from respective engineering domains (e.g. by defining concepts of underdamped,

[31] In Qualitative Process Theory positive influence relation between rate of change of A and value of B is represented as I^+ (A, B)

critical and overdamped). The abstraction of state parameter values w.r.t space is shown in next section.

4.2.2. Domain Specific Constraint Formulation for Mechanical Components Retrieval

Generally for most of the Mechatronics Systems, actuator will be of type mechanical and connected to a power source like motor, engine, pump etc., through complex mechanisms. In such cases, complex BG elements will be required to represent such complex mechanical mechanisms therefore it will be difficult to represent them using the scaling operations introduced in previous sections. However, we can represent such complex mechanism approximately using the Tf or Gy element encapsulating approximate Effort-Flow ratio (of complex mechanism) from power source to power sink or output mechanical actuator port. The framework presented next can then expand that Tf or Gy element while also taking structural constraints into account (relative position and orientation of output port of power source and actuator).

In this section we will look at how to use abstract parameter value space in mechanical domain with semi-qualitative constraints to deduce approximate solution with constraint satisfaction or logical satisfaction techniques, called "semi-qualitative kinematic framework" from here on. The abstraction is used to reduce the parameter value space and to improve decidability of satisfaction algorithm such that it can prove if a design exist for given constraints and initial conditions (input and output link states as shown in Figure 4-7 Figure 4-9).

The semi-qualitative kinematic framework for building rigid body kinematic chains determines number of intermediate links, link size and connection type between the links while satisfying the position and relative velocity constraints between input and output links. Generic mechanical components can then be chosen based on the connection type (which can be either joint or constraint types from Simmechanics[32] library of Simulink because it is general enough to represent all type of kinematics) and their proportional scaling factor (e.g. gear ratio) is determined using the size of links.

The initial or stating state of problem space in terms of input and output links is shown in Figure 4-7 along with description of some of the constraints and objectives.

[32] https://uk.mathworks.com/help/physmod/sm/multibody-systems.html

- Objective: Find sequence of motion states and properties that satisfy semi-qualitative geometrical and kinematic constraints.
- Properties to be found: # of links, lengths, port connectivity etc.
- Constraints to satisfy: velocity ratio between link A and B, total length, port type compatibility, motion constraints on port types etc.

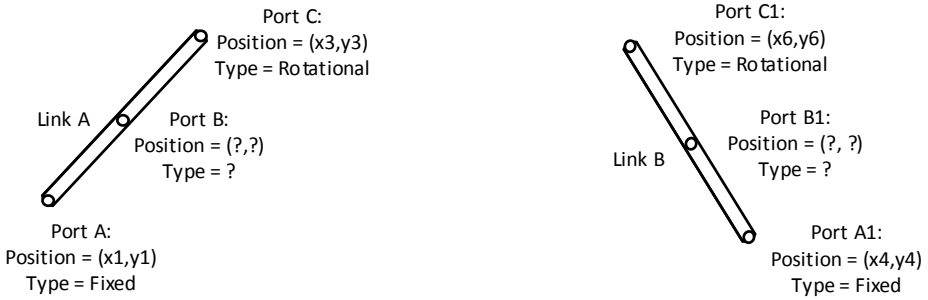


Figure 4-7 Summary of main elements of semi-qualitative kinematic framework and showing 2 initial links with specified positions and types of for two ports.

The Euclidean plane can be divided into quarters and then sectors by using polar coordinates. These quarters and sectors needs to be relatively order for definition of states of the link (i.e. by defining relative position of its three ports). Here sectors are ordered in counter clockwise directions with each sector representing 30^0 interval. These sectors enable resolution of local x and y coordinates (i.e. x' and y') of ports on each side of the link. The resolution is done by using inequalities against ½ length (L) of the link e.g. in 1st quarter at 30^0 the $y' = L \times \sin(30^0)$ therefore $y' < ½ L$ in sector 0 and $x' < ½ L$ in sector 2 and x', $y' > ½ L$ in sector 1. These resolved local coordinates can then be either subtracted or added in linear fashion depending on the relative orientation of two subsequent links to calculate global port positions (x and y). This addition/subtraction is implicitly encoded in the inequalities between quarters e.g. the port's *x* coordinate in right half plane is always greater than when same port is in left half plane in any motion state.

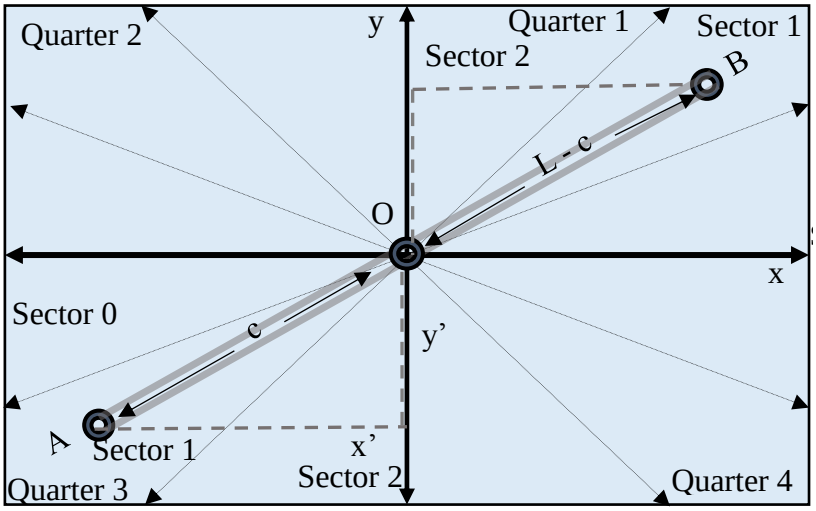


Figure 4-8 Illustration of link's spatial attributes with respect to the cardinal directions and sectors in each quarter.

As shown in Figure 4-8, a link can have three ports with two at terminals and 1 in between (shown in center in figure but it is not constraint to be at centre to give different pivot positions). These ports are divided into four types Fixed, Rotational, Translational and Rotational-Translational for assigning different motion types to these port types so as to enable qualitative kinematics simulation of link through consistent transition between kinematic states (orientation and position). Rotational port can have clockwise and anticlockwise rotational motion and this port can only be assigned to links which have Rotational ports on both sides or if it has a Fixed port on one of its side. The Translational port can only be assigned to links which have Translational port and no fixed port. This port can have relative linear motion (w.r.t links connected to it) in one of the six cardinal direction (North/ positive y, South/negative y, North-West (positive x & y) etc.) e.g. "Tx" motion means translation in increasing x direction while keeping y constant w.r.t connected link. Lastly, Fixed port cannot have any motion and Rotational-Translational port can attain any of motion types but only one in each state e.g. translation followed by rotation or followed by other translation and then rotation. These ports also provide interface for connecting with other links as well (first and last link is constraint to only connect with one of its port whereas others can be connect with two of their ports, therefore every link is constraint to connect to at least one other link).

Below are some of the constraints defining the basic vocabulary required to define more complex concepts and relations (defining different type of joints between links) of this framework:

Kinematic state variables:

Global position of port = (x, y), Local position of link's ports with respect to link's centre is = (x', y') and link's pose or orientation = (quarter, sector)

Motion is determined based on change of these variable during each state transition. Consistent transition between states (position/orientation) of links is maintained by using following axioms on the above state variables:

a) Generic axioms applicable across all port types

i) Global coordinate values of two terminal ports are distinguished by following inequalities comparing their quarter positions in right and left half plane: -

For two different terminal ports p1 and p2 of each link with global position = (x1, y1) and (x2, y2)

- quarter(1, p1) and quarter(3, p2) => x1 > x2 and y1 >= y2
- quarter(2, p1) and quarter(4, p2) => x1 <= x2 and y1 > y2

where quarter(1, p1) means that p1 is in quarter1.

These constraints also defines coordinates directionality i.e. increasing and decreasing coordinate values. Whereas when link is vertical upright (i.e. local coordinate x'=0) then it can only be in quarter 2 and 4, similarly when link is horizontal (i.e. local coordinate y'=0) then it can only be in quarter 1 and 3. This is made more explicit in constraint v).

ii) Two terminal ports of the link are constrained to be in same sector and in diagonally opposite quarters on x-y plane for all *s* states: sector(s, p1) = sector(s, p2) and quarter(s, p2) = next(next(quarter(s, p1))),

where next predicate points to next quarter in counter clockwise ordered list of quarters. 3rd port in between two terminal ports assume quarter and sector of port in left hand plane.

iii) Orientation of the link in each of the 3 sectors is defined by applying following constraints on x', y' local coordinates. For quarter 1:

- x'1 > x'2 > x'3 and y'1 < y'2 <y'3,

and in quarter 2:

- x'1< x'2 < x'3 and y'1 < y'2 < y'3.

Sectors of quarter 3 follows quarter 1 constraints and quarter 4 follows quarter 2 constraints.

Where x'1, x'2, x'3 ($\in \mathbb{Z}$) and y'1, y'2 and y'3 ($\in \mathbb{Z}$) are local coordinate values in sector 0, 1 and 2 respectively.

iv) Local variables for each terminal port are defined by using inequalities against the length AO = *c* and OB = L-*c* (see Figure 4-8), where L is link's length. Following constraints are shown for each quarter on right and left hand half planes whereas other two follows similarly;

Following constraints defines local variables for all terminal ports p in quarter 1 and sector 0 and 2: -

- Sector(0, p) and quarter (1, p) => x'> 0.5 L - c && y' < 0.5 L - c
- Sector(2, p) and quarter(1, p) => x'< 0.5 L - c && y'> 0.5 L - c

For all terminal ports p in quarter 2 and sector 0 and 2: -

- Sector(0, p) and quarter(2, p) => x'< 0.5 c && y' > 0.5 c
- Sector(2, p) and quarter(2, p) => x'> 0.5 c && y'< 0.5 c

Local coordinate of ports in quarter 3 and quarter 4 are then defined by combination of above constraints and link's length constraint below,

For all links in all states with local coordinates x'1, y'1 and x'2, y'2 as well as global coordinates x1, y1 and x2, y2 for terminal ports p1 and p2 respectively,

- $(L-c)^2 \leq \{(x'1)^2 + (y'1)^2\}$ following triangle inequality
- $c^2 \leq \{(x'2)^2 + (y'2)^2\}$
- x'1/y'1 = x'2/y'2, ratio between two needs to be same for link to be straight

With cx, cy as middle port's global coordinates we have for all links and in all states,

x'1 = x1 – cx, y'1 = y1 – cy similarly constraints exist for terminal port 2

For all terminal ports p in sector 1 of quarter 1 and 4:

- Sector(1, p) and quarter(1, p) => x'$\geq$ 0.5 L -c && y'$\geq$ 0.5 L -c

For all terminal ports p in sector 1 of quarter 2 and 3:

- Sector(1, p) and quarter(2, p) => x'$\geq$ 0.5 c && y'$\geq$ 0.5 c

v) Boundaries between quarters are defined by relating global and local coordinates in sector 2 and 0 of each quarter when link is either along x-axis or y-axis. Boundary has to belong to one of the quarter and its sector therefore following constraints are given

Boundary between quarter 1 and quarter 4 always belongs to quarter 1:

- y1 = y2 and x'1 = L-c => sector(0, p1) and quarter(1, p1),

where y1 and y2 are global coordinates of terminal ports p1 in right half plane and p2 in left half plane on same link.

Boundary between quarter 1 and quarter 2 always belongs to quarter 2:-

- x1 = x2 and y'1 = L => sector(0, p1) and quarter(2, p1)

where x1 and x2 are global coordinates of terminal ports p1 and p2 on same link.

Similarly, boundary between quarter 2 and 3 will belong to quarter 3 due to ii) constraints.

b) Port type specific axioms enables definition of rotational and linear motion. These axioms uses change in global coordinates to deduce change in local coordinates and then to infer change in sector. Directionality of sector transition enable conclusion of quarter transition. Or more briefly,

Change in x, y <=> change in xd, yd => change in sector => change in quarter => type of motion

i) Definition of clockwise and counter-clockwise rotational motion is based on change in local coordinates within each quarter between two successive states. This position change within the quarters of a port requires that none of the global coordinates of the two ports are equal or none of the local coordinates are equal to link's partial lengths c and L-c:

For all terminal ports p1 with local coordinates in state 1= (x'1, y'1) and in state 2 = (x'2, y'2)

- In quarter 1, x'1 > x'2 and y'1 < y'2 => CCW and x'1 < x'2 and y'1 > y'2 => CW
- In quarter 2, x'1 < x'2 and y'1 > y'2 => CCW and x'1 > x'2 and y'1 < y'2 => CW

By symmetry of quarter 1 to quarter 3 and quarter 2 to quarter 4 following constraints will be implied from above 2.

- In quarter 3, x'1 > x'2 and y'1 < y'2 => CCW and x'1 < x'2 and y'1 > y'2 => CW
- In quarter 4, x'1 < x'2 and y'1 > y'2 => CCW and x'1 > x'2 and y'1 < y'2 => CW

Defining rotational motion in terms of local coordinates allows translation and rotation (without any rotation of link around any of its port) to be applied to the centre of the link without affecting its local coordinate values or its independent motion as driver joint. Or in other words,

it allows rotational and translational transformation to be applied to a joint's coordinates with respect to its preceding joint.

ii) Transition between sectors of same or different quarters is based on the change in local variables. Following constraints defines directionality of such transition due to different types of rotational motion.

For all rotational and rotational-translational port p1 in current state *s* and next state *s'* and sector *sc* in current state:-

- sc =1 and sector(s', p1) = sector(s, p1) or sector(s', p1) = next(sector(s, p1)) => motion(s, p1, CCW)
- sc =1 and sector(s', p1) = sector(s, p1) or sector(s', p1) = prev(sector(s, p1)) => motion(s, p1, CW)
- sc = 2 and sector(s', p1) = sector(s, p1) or sector(s', p1) = 0 => motion(s, p1, CCW)
- sc = 0 and sector(s', p1) = sector(s, p1) or sector(s', p1) = 2 => motion(s, p1, CW)

iii) Transition between adjacent quarters only occurs when a port is in sector either 0 or 2 and when one of the two global coordinates of two terminal ports either become equal to each other or has certain minimum difference:

For all rotational and rotational-translational port p1 in current state *s* and next state *s'* and sector *sc* in current state:

- next(q, q1) and quarter(s', p) = q1 and sector(s', p) = 0 and quarter(s, p) = q and sector(s, p) = 2 => motion(p, CCW)
- prev(q, q1) and quarter(s', p) = q1 and sector(s', p) = 2 and quarter(s, p) = q and sector(s, p) = 0=> motion(p, CW)

where next() predicate relates two adjacent quarters in counter clockwise direction and prev() does opposite.

"Transition between quarters" axioms also include conditions checking, whether or not, one of the global coordinates (i.e. x or y) of two terminal ports are equal for determining boundary between the quarters:-

For ports p1 and p2 on same link with positions (x1, y1) and (x2, y2) respectively,

- quarter(s', p) = 2 and sector(s', p) = 0 and quarter(s, p) = 1 and sector(s, p) = 2 and x1 - x2 < ϵ => motion(p, CCW)

- quarter(s', p) = 3 and sector(s', p1) = 0 and quarter(s, p) = 2 and sector(s, p) = 2 and y1 - y2 < ϵ => motion(p, CCW)
- quarter(s', p) = 4 and sector(s', p1) = 0 and quarter(s, p) = 3 and sector(s, p) = 2 and x1 - x2 < ϵ => motion(p, CCW)
- quarter(s', p) = 1 and sector(s', p1) = 0 and quarter(s, p) = 4 and sector(s, p) = 2 and y1 - y2 < ϵ => motion(p, CCW)

where ϵ is any small integer. For definition of CW under quarter transition same conditions in opposite manner are used and also exact match is used between global coordinates i.e. x1 = x2 or y1 = y2.

ii) Definition of Translational motions

A link with Translational or Rotational-Translational ports on both sides can translate in one of the eight directions i.e. x, nx, y, ny, xy, xny, nxy and nxny, where nx = negative x direction. It also preserves its orientation (quarter, sector) by keeping difference between local coordinates of two terminal ports same across successive states as both ports translate in same direction (in terms of increasing/decreasing x and y) and with same magnitude. Following constraint applies across all Translational or Rotational-Translational terminal ports having translational motion with local coordinates (x', y') and (x'1, y'1) in current state *s* and next state *s'* respectively. motion(p, Translation) => x'1 = x' && y'1 = y' && quarter(s, p) = quarter(s', p) && sector(s, p) = sector(s',p)

For centre port cp with global position (cx, cy) in current state and (cx1, cy1) in next state, translational motion in x and nxny direction can be defined as;

- motion(s, cp, Tx) => cx1 > cx and cy = cy1
- motion(s, cp, Tnxny) => cx1 < cx and cy < cy1

whereas definition for other translational directions follows in similar way. Please note that motion of link revolving around port of some other link is not taken into consideration when defining constraints above.

Different types of motions defined can now be assigned to different types of ports and combination of these ports can be used to define different type of connections. Connection types determines relative motion between two links based type of ports connected e.g. A fixed type of port can be connected to all type of ports but in case of linear coupling connection the two connected ports does not keep same global coordinates in all states whereas in all other connection types they does.

A connection can be active (as a driver) or passive. Passive connections are Pin (or Pin-Bearing) joint, Linear Coupling, Shaft Coupling, Weld joint and Gear constraint. Pins can be used for connecting Rotational ports, Rotational with Rotational-Translational ports and Translational with Rotational-Translational ports. Pin connection requires two links to have relative rotational motion. Translational ports can only be connected by using Linear Coupling or it can be connected with Rotational-Translational port by using a Pin. Gear Constraints can only be used between those two links which have a Fixed port at the middle or Rotational ports on both sides (when middle is fixed). Shaft Coupling connects middle ports of two links and Weld joint connects Fixed ports together. Only one port on a link is allowed to have a Fixed port and also if two links are connected with weld connection then no relative angular or linear motion is allowed.

Angular Driver called Motor Shaft and Linear Driver called Linear Piston can be used as motion source in the kinematic chain. A Motor shaft connection can be placed in between a Fixed port of the Ground and Rotational port of first or last link in the chain whereas the Linear Driver connection can be placed in between a Fixed port of the Ground and Translational port of first or last link in the chain.

This framework in summary includes the concepts or vocabulary necessary to capture the geometric relationships between stationary and non-stationary states of ports on same or different links connected by the joints. These concepts or vocabulary can be used to define new joints or constraints.

4.3. Results and Discussion

Only 2-D semi-qualitative kinematic framework is implemented as retrieval of domain specific descriptive or simulation model using meta-models and rules has been demonstrated extensively (see section 2.3.1.) However, use of BG methodology for constructing function flows by using causal orientation based specialisations (eTF, fTF etc.) and then extending it with use of kinematic constraints has not been done before based on literature review. Framework was implemented using Alloy Analyzer[33] and declarative code is given in appendix section 8.2. Figure 4-9 shows multi-linkage mechanism designed using requirements of having one fixed port in each of first and last link of chain as well as those fixed ports should have same global y coordinates and they shouldn't be connected to any other port. Connection between middle (Cport) and terminal (Tport) are not required as well as no connection of type gear is required. Predicate (named "ex") expressing these requirement is given below, where

[33] https://alloytools.org

plord is the name of ordered list of 2-D planes over which links are residing and one link can only reside on one such plane. Where #Tport.connects > 2 means that there should be at least more than 2 ports involved in connections.

pred ex{ Port.connection & Gear = none && Fixed & rel/dom[connects] = none && #Tport.connects>2 && Tport.connects & Cport =none && one(plord/first.over.tport & Fixed) && one(plord/last.over.tport & Fixed) && ord/first.y[(plord/first.over.tport & Fixed)] = ord/first.y[(plord/last.over.tport & Fixed)] }

run ex for 8 but 0..127 Int, 3 Body1d, 9 Port,2 State, 3 Plane, 5 seq, 4 Dir

This predicate ("ex") is only tested for at most 3 Bodies or links and 2 states and it took 28.414 hours to run on octa-core Intel xeon e3-1505m v5 set at 2.80 GHz. Although the CPU usage for most of the part stayed between 25-35%. It is only possible to use integers in a limited range for values of global & local coordinates, velocity and for other numerical variables in Alloy Analyzer therefore it is not very useful to demonstrate inexact/approximate velocity proportion (between first and last link) requirement satisfaction using limited precision. However it is possible to formulate velocity proportion target using number of sectors that ports of first and last link relatively travels. But sector motion constraints were not used in this experiment as it requires other related constraints to be used which increases solver run time significantly and the objective framed in the predicate can still demonstrate automatic selection of properties such as port connectivity, connection selection, length determination, motion type allocation etc.

Some insights to develop a methodological approach to abstract a domain with strong or known domain theory:

- Symmetry between value space needs to identified and exploited to reduce number of constraints as well as to reduce number of qualitative values (quarter 1, sector 1 etc.).
- Most primitive and invariant constraint of a domain should be identified and based on which the domain specific concepts be defined (e.g. length constraint was identified and used as backbone for other concepts such as quarter, velocity etc.).
- Complex and reusable constraints should be given alias or name for easy reuse (e.g. angular velocity represents relationship between velocity and partial length and can be used for defining relative motion between two links having pin joint).

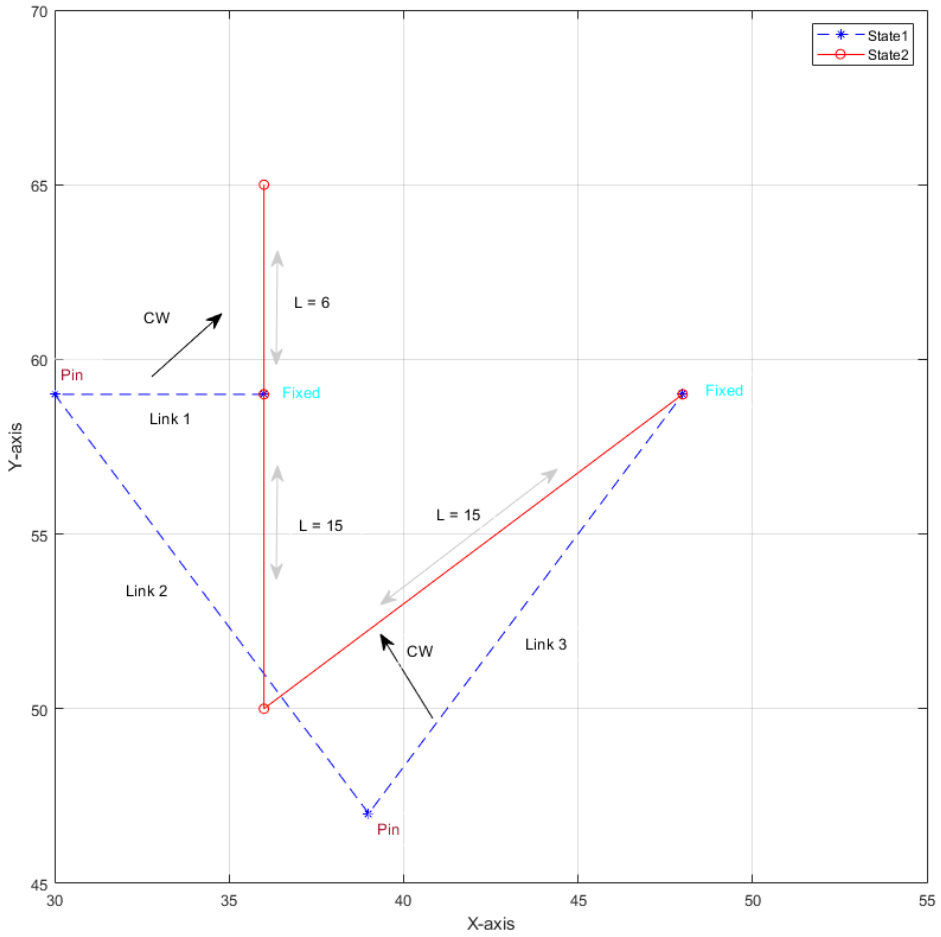


Figure 4-9 Closed loop 4 bar linkage design without specifying required velocity and port positions

Chapter 5 Behaviour Based Design Storage and Retrieval

This chapter presents a novel method of using Auto-encoder embedding to represent designs using their behaviour data obtained under a particular use case scenario. This chapter will also analyse the choice of different clustering algorithms and how well they cluster designs w.r.t topology and configurations of that topology to enable retrieval of similar known design using different criteria.

A system can have non-functional requirements given in operational requirement. These requirements are decomposed and allocated to logical sub-systems or functions conceptualised in a logical behaviour architecture model representing ideal expected behaviours due to interaction between internal system's functions or their interactions with external entities in different use case scenarios. These behaviours can be represented as traces or trajectories (w.r.t time or space) that can result due to the combination of different input profiles and state of other functions depending upon use case scenarios. On the other hand, retrieved candidate descriptive physical designs substituting logical subsystems (using only port type and other symbolic information as described in section 4.2.) can be further shortlisted by determining if they have similar behaviour to the ideal behaviour allocated to logical sub-systems which they substituted. Solution designs (corresponding to descriptive designs) can be represented using numerical encoding of a neural network by using simulation behaviours generated by design's components under various input profiles (applied to substituted logical subsystem). In the approach presented in this chapter we focus on retrieving or completing partial designs which exhibit similar behaviour to that of required partial behaviour (i.e. required parameter behaviour of source and load) while operating under same SAD.

Following assumptions are made:

1. Designs with similar topology and properties exhibit similar characteristic behaviours (from similar type of components) under same input conditions and therefore will be closer to each other in latent space.
2. If all but characteristic behaviours of one of the design in a cluster is omitted then that design will still be clustered with same neighbourhood designs as before when all of its components behaviour was given.

The emergence of world of generative modelling using GAN [72] and VAE [63] has open up the possibility to explore new horizons in the field of Mechatronic system modelling. There are now tools available which have potential to learn association between system behaviour patterns, physical structural arrangement and structural properties giving rise to those behaviours. Generalisation over such associations can enable inference of one when other is

either fully or partially observable. Such capability alludes to the possibility of performing design alternatives generation and selection or design optimisation in one go.

This chapter demonstrates unsupervised feature representation learning capability to represent similarity between known designs as well as between know-unknown designs by using distance metric on the design (full or partial) behaviours encoded in compressed Euclidean space where structurally similar design sit close to each other. In essence we test the hypothesis that "*learning hidden factor of variations underlying observed behaviours of designs leads to clustering of structurally similar designs*". These hidden factors of variations are captured by the use of latent representations learnt by Maximum Mean Discrepancy Variational Auto-Encoder (MMD-VAE).

It is also assumed that the behaviour of system can be entirely captured in Effort (Voltage, Force etc.) and Flow (Current, Velocity etc.) variables conceptualised by Bond Graph Methodology for modelling Mechatronic system. For training and inference, it is required to feed captured signals to a Machine learning algorithm in a format that suits the chosen neural network architecture as well as preserve the distinct characteristics of the input data (e.g. causality of time series, more details in section 5.4.)

5.1. Motivational Example

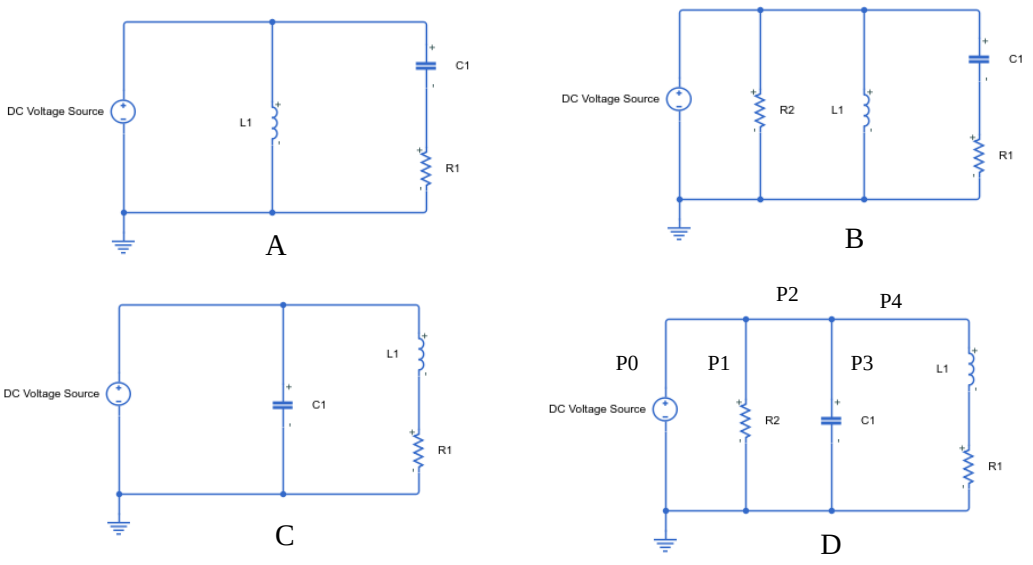


Figure 5-1 Example electronic circuits with different configurations of components

Figure 5-1 shows two pairs of designs a, b and designs c, d which differs from each other structurally due to difference in position of inductor and capacitor where designs within each pair are different because of addition of another resistor. We assume same values for C1, R1 and L1 in all designs as well as same value for R2s in both b and d.

When all of these designs are powered by same input source pairs a, b and c, d produce different behaviour (current and voltage) across each of its components because of difference in topology. The proposed approach determines correct pairing or cluster for each of the design even though the addition of a resistor changes behaviour of rest of circuit components in terms of rise time, amplitude of oscillations, steady state amplitude of voltage or current etc. Also if we have multiple configurations of each topology with different values for resistor R2 then configurations of each topology are grouped separately from configurations of other topologies as well as distance between configurations of topology pair (a, b) is less than compared to distance to configurations in topology pair (c, d). Representation of embedding in 2-D space of 4 topologies with 2 configurations each is shown in Figure 5-2.

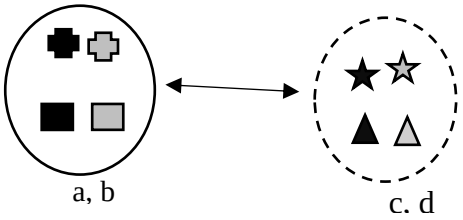


Figure 5-2 Different symbol shapes representing different topologies with different colour shade representing configuration

If suppose we want to find a design which exhibit behaviour matching required output behaviour of a logical subsystem across one of its component when required input is applied. As we only know required I/O behaviour then this can thought of partial data representation of required design i.e. only expected current or voltage signal across time forms the partial design. This partial design or group of signals then can be embedded in latent space along with signal groups from all of the four known designs. Nearest known design can be retrieved based on least metric distance between embedded signal groups of known and partial design. This nearest design(s) can be considered as a candidate design(s) which can be put through further screening.

Assumption is that path with similar value configuration (over non-zero property types) will be closer than others under same input conditions.

5.2. Method

As discussed in section 2.4. , dimensionality reduction with unsupervised training neural networks helps to bring similar data examples closer to each other after applying deterministic or in-deterministic transformation(s) to reduce the input data dimension. Auto-encoder neural network are often used for such dimensionality reduction tasks. They are usually comprised of two identical neural networks namely an encoder network and a decoder network. The encoder network compresses the input into a low dimensional representation (often called embedding space, latent code, hidden state etc.) whereas a decoder learns the inverse transformation to that of encoder.

Recently developed generative modelling techniques such as Variational auto-encoder, GAN (Generative Adversarial Networks) [72] and their refinements and derivatives, learns embedding or latent space whose each dimension represents a different factor of variation e.g. when applied on face dataset each dimension can represent face expression, face rotation, etc.

Extension of GAN has also been applied on time series data and it's been observed that the embedded signals vary in their shape when there corresponding latent points are linearly interpolated [73] however it's not been shown whether or not each latent dimension captures particular aspect of the signal (e.g. number of oscillations, peak to peak magnitude etc.).

Because we are dealing with time dependant causal data it is best to use neural network which take into account temporal dependencies. Following are some of the widely used NN architectures used for temporal relation modelling to predict or classify time series or signals:

1. Recurrent neural network

2. Long Term Short Term Memory networks

3. Convolution with causal padding [74]

Whereas following neural network architectures are used to generate new time series by learning the data distributions:

4. Wavenet [75]

5. Pixel-RNN [76]

6. RCGAN [73]

Apart from neural network approaches, there are other Machine learning approaches e.g. Gaussian processes which are also suitable for temporal relation modelling (because of their assumption of sequential order in input points) and require less amount of data for training

compared to neural network to learn. Deep Gaussian process can approximate non-Gaussian distributions (which result when multiple signals belonging to behaviour data of a design are used as input) using variational inference [77] and does not uses explicit latent code however each layer is dependent on induced points (sample of output points) of previous layers which can be considered as sampled latent points.

5.3. Representing Design Behaviour Data for Applying Machine Learning

As we are mainly dealing with multivariate time series or causal signals, it is important to match the format of input data to selected network architecture as well as the input should embody the structural relationship as much as possible. The chosen strategy uses 2-D input format where signals from different path (containing components with either same flow or same effort) are stacked along the rows and signals belonging to same path are stacked along the columns. For electrical circuits, each path (with same flow) can contain 3 different types of components without permutations (R, L, C) with 4 voltage points across them and same current following through them giving in total 5 signals in each path e.g. the circuit *d* in Figure 5-1, has 5 different paths based on 5 different values of DC voltage source to ground current (e.g. P0..P4 are the current paths labelled in schematic). The 2-D input format used for circuit *d* can be seen in Table 5-1where each cell corresponds to one signal of 1000 time steps. Same input or output effort signal is repeated in case if there are less than 3 different components present in one path e.g. row 2 of Table 5-1 has only 1 capacitor in path P3 therefore the table is filled by assuming $E_L = E_C$, $E_{Cout} = E_R$ and $E_{Rout} = E_{Cout}$, where E_L is absolute inductor voltage, E_{Cout} is capacitor voltage measured at negative end, E_R is resistor voltage, F is current of respective path etc.

Non-Zero Paths	**Components**	**E_L**	**E_C**	**E_R**	**E_{Rout}**	**F**
P4	L1 R1	…	…	…	…	…
P3	C1	…	…	…	…	…
P1	R2	…	…	…	…	…

Table 5-1 Input data format used for representing behaviour data of electric circuits (row order is arbitrary)

Therefore the system behaviour data comprises of effort signals across all the components and different flow signal of each path connecting components. The duration over which all signals are captured as well as the sampling time (therefore number of signal timesteps) is determined by the maximum and minimum time constant as well as bandwidth of a system under respective system's engineering class and SAD (e.g. time constant of a mechanical system is significantly bigger than an electrical system as well as signal frequency ranges differs between power electronics vs communication networks).

5.3.1. Train and Test Data for Experiments

The experiments are conducted using electric circuit data as they are easier to construct (i.e. only contain 3 main properties) and can exhibit behaviour characteristics common to all Mechatronic designs (e.g. damped, underdamped and overdamped characteristics of continuous dynamic system). The test data set used in the experiments is composed of data gathered from simulation of 15 circuit topologies (manually designed) 13 of which are simulated with both voltage and current input source whereas parallel RLC and series RLC are simulated with current and voltage source respectively. 11 circuit topologies uses 4 components (2 resistors, an inductor and a capacitor) with capacitor, inductor and one of the resistor having same value across circuits (C = 1e-6 F, L = 1e-3 H and R2 = 100 Ohm) and their configurations. Whereas 4 topologies only has 3 components with R2 = 0 Ohm. Each circuit then has 6 configurations based on different resistor values (only R1's value is varied from 1 to 36 Ohm with 7 interval difference). The training set is formed of 100 randomly generated circuit topologies (by using a common circuit format given in Figure 5-3) with only 4 of components (acting as primary components) having value (randomly set within a min max value[34]) that can impact circuit behaviour. Similar to test data generation, 6 configurations (in each configuration all component values were randomly chosen) of each of the 100 topologies with both current and voltage input source were generated i.e. 600 X 2 circuit simulations. Each placeholder subsystem of Figure 5-3 contains 3 parallel branches which can have at-most 3 series R-L-C components. Only 4 components out of all branches of placeholder subsystems are activated and randomly assigned value based on pre-selected min max values whereas rest of components are only activated to complete the circuit (acting as supporting components). These supporting components are given either very huge (e.g. resistance set to very high to make the circuit open at that branch) or very small value (e.g. resistance set to low in subsystem if it doesn't have any primary component to make circuit complete if subsystem 4 has a primary component) based on type and location of that supporting component. The voltage source is kept at 1 V and current source at 1 A in all circuits. Sampling time was set at 1e-5 seconds and simulation time at 1e-2 seconds based on time constants of parallel and series RLC circuits in test data. Each row containing signals corresponding to one path of a circuit configuration is fed as a separate example when training and testing.

From here on, the circuits will be named based on format "Source type component1 component2component3" i.e. component 1 is parallel to component 2 and component 3 which are in turn in series with each other e.g. the *d* circuit in example can be named as Vs R2 C LR.

[34] Inductor = [1e-6…1e-1] H, Capacitor = [1e-9…1e-3] F, Resistor = [1…100] Ω

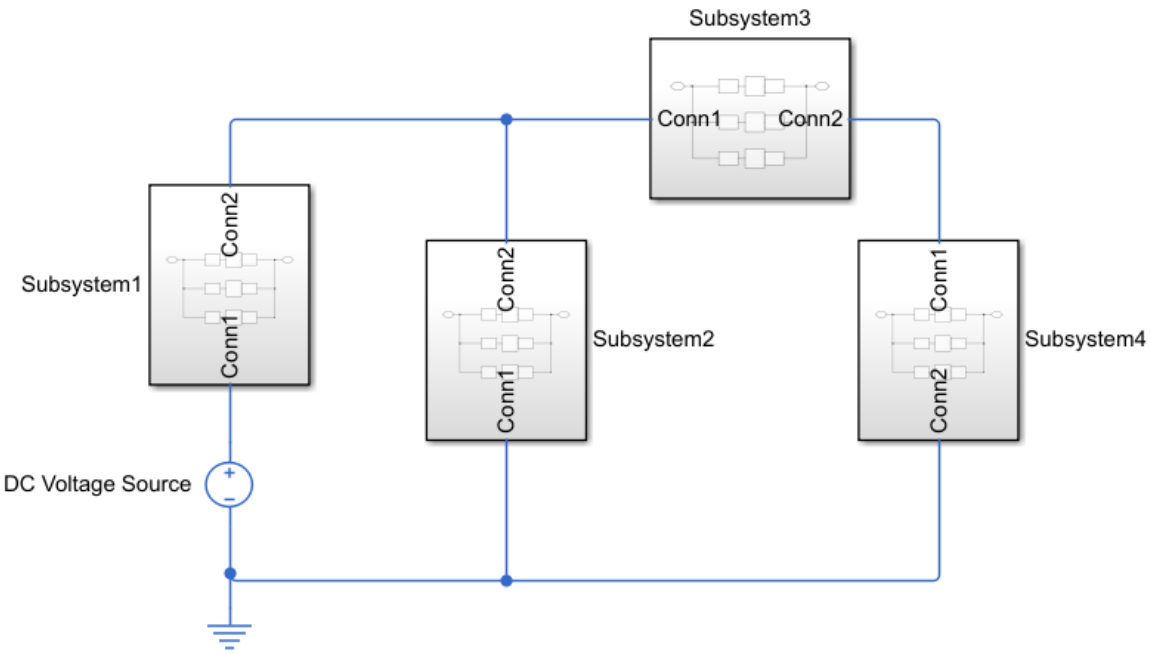


Figure 5-3 Common circuit format (can be with voltage or current source) having 4 subsystems with 3 parallel paths with each path containing placeholder or dormant series RLC components

5.4. Clustering of Electric Circuits

Objective of the following experiments is to identify algorithm which can cluster together circuits with some pattern e.g. cluster of circuits sharing same source type or cluster of circuits having same topology with only a difference of one component. Similarity data related to only 1-11 test circuits (shown in section 8.3. of appendix with their index) is considered here because those circuits can be used to form clusters with three main characteristics i.e. cluster of circuits with similar topologies (with difference of one parallel R2 resistor) and similar source type, cluster based on only topology similarity and cluster of circuits based on source type similarity.

I. Clusters sharing similar topology with difference of one resistor and source type are:

(1 – Vs R2 L RC, 4 – Vs L RC), (3 – Vs C RL, 5 – Vs R2 C RL), (7 – Is C RL, 9 – Is R2 C RL), (8 – Is L RC, 11 – Is R2 L RC)

II. Clusters sharing similar topology with difference of one resistor only are:

(1 – Vs R2 L RC, 4 – Vs L RC, 8 – Is L RC, 11 – Is R2 L RC), (3 – Vs C RL, 5 – Vs R2 C RL, 7 – Is C RL, 9 – Is R2 C RL) and (6 – Vs R2 RLC, 10 – Is R2 RLC)

III. Clusters sharing similar topology (resistor difference and L/C swapped) and source type:

(1 – Vs R2 L RC, 4 – Vs L RC, 3 – Vs C RL, 5 – Vs R2 C RL), (8 – Is L RC, 11 – Is R2 L RC, 7 – Is C RL, 9 – Is R2 C RL)

Many methods apart from using Neural Network exist to perform clustering of multivariate time series data e.g. using maximum correlation coefficients, k-means clustering etc. But benchmarking data shown in Table 5-2 to Table 5-9 shows that such simple methods fall short of achieving the required clustering when considering the dimensionality of input data. The benchmarking was performed by different ways of normalising data, using different distance metrics, by using different number of clusters (in case of k-means) and different methods of summarising metrics (i.e. determining circuit to circuit proximity based on all rows of its each configuration).Please note from here on we will refer each topology with different source type as a different circuit.

1) Taking mean of maximum values of normalised cross correlation over all row pairs of configurations of each circuit pair i.e. $Corr_{Ci,\ Cj} = \frac{1}{R*C}\sum_{i=1}^{R*C} max\{cross\ corr(x)\}_{t=0}^{T*J}$, where Corr = circuit to circuit maximum average correlation, C = number of configurations, R is number of rows, T = number of signal samples and J is number of signals in each row. Following table shows that for each target circuit.

Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Descending correlation of target circuit with all circuits	14	17	13	14	13	14	14	14	23	14	14
	17	14	25	17	25	17	17	17	14	17	17
	23	12	21	23	21	12	23	12	17	23	12
	12	23	27	12	27	23	12	6	6	12	23
	6	6	17	6	17	6	6	23	12	6	6

Table 5-2 Circuit to circuit maximum correlation averaged over all of their configurations

2) By taking RMS average (instead of using maximum value) over negative and positive normalised correlation values

Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Descending correlation of target circuit with all circuits	21	21	21	21	21	21	21	21	21	21	21
	13	13	13	13	13	13	13	13	13	13	13
	25	25	25	25	25	25	25	25	25	25	25
	19	19	19	19	19	19	19	19	19	19	19
	27	27	27	27	27	27	27	27	27	27	27

Table 5-3 Circuit to circuit RMS correlation averaged over all of their configurations

No meaningful distance order between target circuits and other circuits can be identified in both experiment 1) and 2).

3) By using k means clustering with 28 clusters and L1 distance metric over non-normalised circuit data. Each circuit was assigned a cluster by taking mode over cluster assignment of all rows and all configurations of each circuit.

Cluster assignment to each circuit											
Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Cluster ID	22	1	22	1	22	1	17	1	17	13	1

Table 5-4 Cluster assignment to each circuit using k-means with L1 distance metric and 28 clusters

Here we identify clusters of circuits (1,3,5), (2,4,6,8,11), (7,9) and (10) with 4,8 & 11 in (2,4,6,8,11), 3 & 5 in (1,3,5) and both of circuits in (7,9) shares similar topologies. But clustering is not consistent i.e. 1 and 4 also have same topology but are not clustered.

And with 9 clusters,

Cluster assignment to each circuit											
Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Cluster ID	1	1	1	1	1	1	5	1	5	1	1

Table 5-5 Cluster assignment to each circuit using k-means with L1 distance metric and 9 clusters

4) Using PCA projected data for k means clustering with L1 distance metric and 28 clusters

Cluster assignment to each circuit											
Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Cluster ID	14	11	1	14	1	1	12	27	4	6	27

Table 5-6 Cluster assignment to each circuit using k-means with L1 distance metric and 28 clusters with PCA projected data

In these results we only have 1 major cluster (3,5,6) .

And with 9 clusters,

Cluster assignment to each circuit											
Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Cluster ID	8	1	8	1	8	1	7	1	6	4	1

Table 5-7 Cluster assignment to each circuit using k-means with L1 distance metric and 9 clusters with PCA projected data

In above table we have clusters of (1,3,5) and(2,4,6,8,11) which is similar to result of experiment 3) without PCA and with 28 clusters.

5) Using k-means clustering on non-normalised input with L2 distance metric and 28 clusters

Cluster assignment to each circuit											
Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Cluster ID	1	1	1	1	1	1	16	1	15	5	1

Table 5-8 Cluster assignment to each circuit using k-means with L2 distance metric and 28 clusters

And with 9 clusters,

Cluster assignment to each circuit											
Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Cluster ID	1	1	1	1	1	1	6	1	6	7	1

Table 5-9 Cluster assignment to each circuit using k-means with L2 distance metric and 9 clusters

None of the results in 4) shows consistent common cluster characteristics across all clusters i.e. clusters of circuits sharing same source type or having same topology with only a difference of one component.

Using other distance metrics (with & w/o PCA), like correlation and cosine, only one common cluster is assigned to all 11 circuits because cluster assignment of none of the rows of each of the circuit (across all of its configurations) receives a distinct majority cluster assignment for that particular circuit.

6) Whereas results in rest of this section shows circuit topologies (1-11 except 2,6 & 10) under same source type (yellow shaded for Voltage and red shaded for Current source) and with only a difference of additional resistor R2 (represented by white font colour and asterisk) nearest to each other. These results are obtained using architecture shown in Figure 5-4 with mean squared loss as reconstruction error and using MMD latent loss[35] to match embedding distribution samples to Gaussian distribution samples. Python code used with Tensorflow library is given in section 8.6. of appendix. Data was scaled either between (0 to 1) or (-1 to 1) across each configuration of a topology or across all configurations of a topology. Distance between circuits is calculated using mean square error between embeddings of configurations (or between set of rows) and then averaging over the error between two circuits i.e. $D_{Ci,\ Cj} = \frac{1}{C}\sum_{i,j=0}^{C*N}\left\|C_i, C_j\right\|_2$, where N is total number of circuits and C is number of configurations.

The signals from paths of each topology configuration are embedded in low dimensional space by using encoder and decoder network shown in Figure 5-4. Appendix section 8.1. shows comparison between performance of different auto-encoder architectures against the 3 different types of clustering strategies. This network applies three 1-D causal convolutions (kernel size,

[35] https://github.com/ShengjiaZhao/MMD-Variational-Autoencoder

stride (S), number of filter (F), output activation) on each signal separately along its time dimension then applies a layer of linear Dense on concatenated intermediate representation of all the signals to generate the embedding. The reconstruction of embedding is done using decoder made up of combination of 1-D causal convolution and 2-D transpose convolution layers (with same padding), where number of convolution layers were always kept same as encoder with number of transpose layers depending on number of stride = 2 used in encoder. L2 regularisation with coefficient 0.001 was used on all layers except last Dense layer delivering output and fixed 1e-4 learning rate with 72 batch size was used.

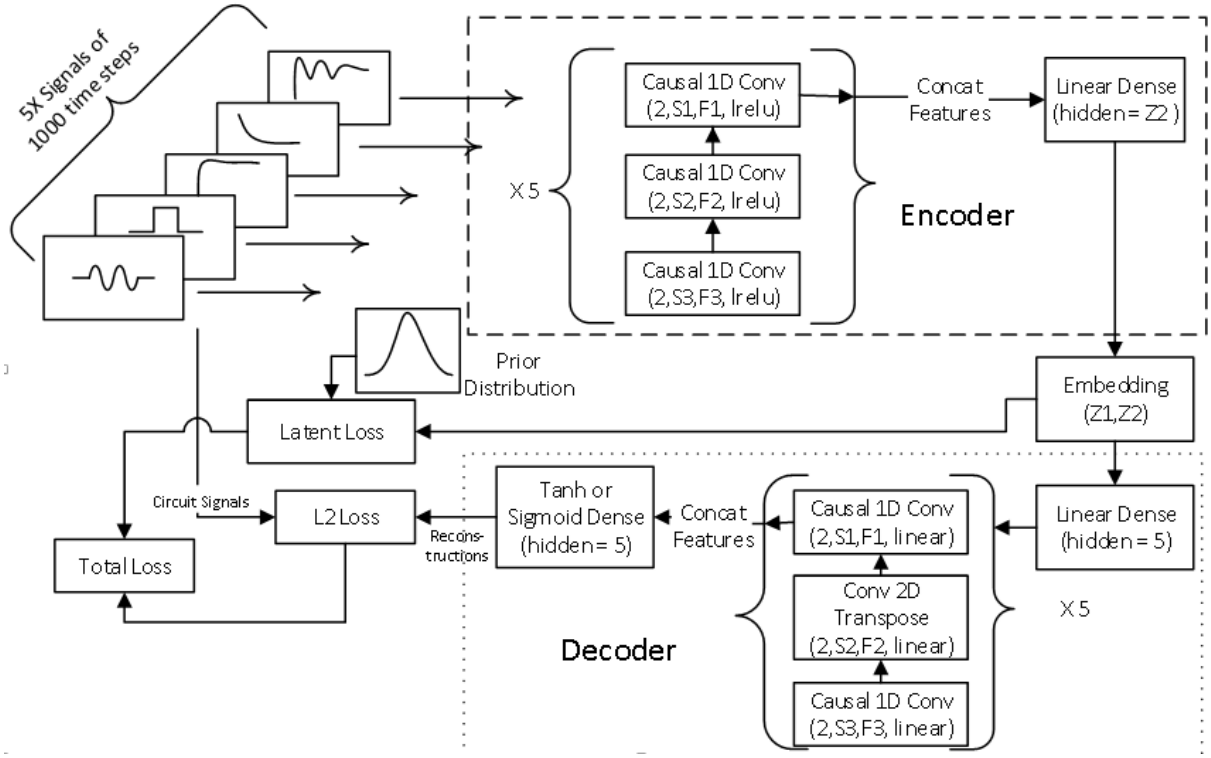


Figure 5-4 Architecture of MMD VAE Auto-encoder used for clustering similar circuit

I. MMD configuration 1) Z1 = 500 and Z2 = 3 embedding dimension MMD-VAE with MMD loss applied across signal dimension for each time step, 0-1 normalisation for each configuration, F1 = 128, F2 = 64, F3 = 32, S1=S3=1, S2 = 2 and sigmoid dense at output.

Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Ascending distance of target circuit to all circuits	1	2	3	4	5	6	7	8	9	10	11
	4*	13	5*	1*	3*	8	9*	11*	7*	5	8*
	17	20	16	17	10	11	27	6	26	1	6
	16	26	4	16	1	27	26	27	10	13	27
	3	1	17	3	16	14	25	14	25	17	14
	5	4	1	5	4	26	14	26	27	16	26

Voltage Source | Current Source | * Circuit with difference of Resistor R2

Table 5-10 Clustering results of MMD-VAE configuration 1 @10k epochs using pair wise configuration to configuration distance.

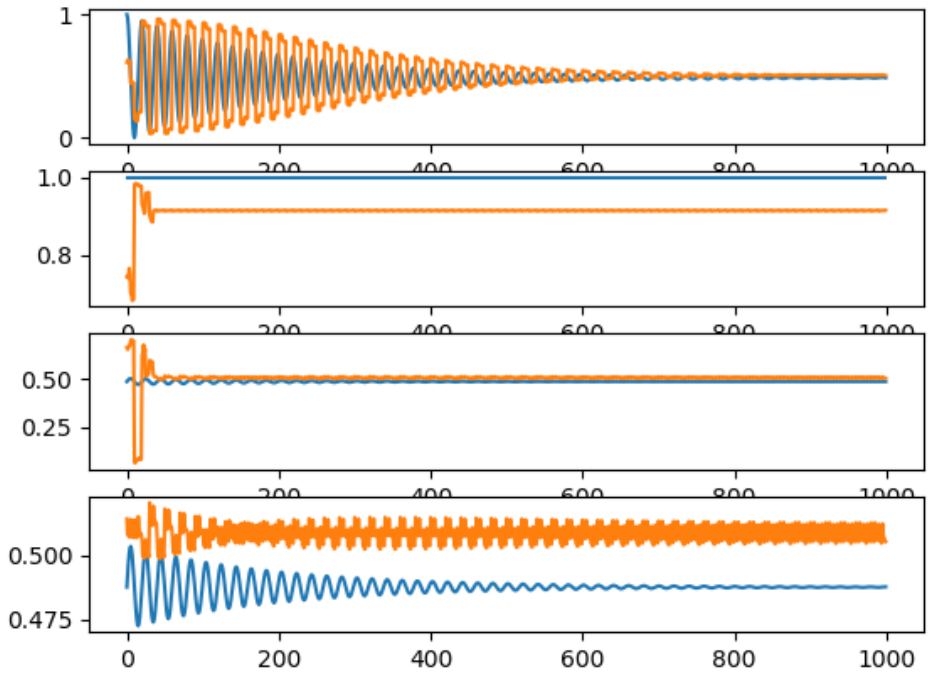


Figure 5-5 Reconstructions (in Red) of (top to bottom) Voltage and Current of R,L & C in Vs R2 RLC circuit's 1st configuration as a result of MMD-VAE configuration 1

Ordered Circuit Configurations w.r.t C1V1	**L**	Ordered Circuit Configurations w.r.t C1V1	**RC**
C1V1	1	C1V1	2
C4V1	217	C4V1	218
C1V2	13	C1V2	14
C1V3	25	C4V2	230
C1V4	37	C1V3	26
C1V5	49	C4V3	242
C1V6	61	C1V4	38
C4V2	229	C4V4	254
C4V3	241	C1V5	50
C4V4	253	C4V5	266
C4V5	265	C1V6	62
C4V6	277	C4V6	278

Table 5-11 Row to row distance of MMD configuration 1 based on 12 rows (1 + 12x L rows and Row 2 + 12x RC rows) of each configuration (V1 - V6) of circuit Vs R2 L RC (1) and circuit Vs L RC (C4) with "x" multiple between (1,6)

II. MMD-VAE configuration 2) Z1 = 500 and Z2 = 3, acting as Auto-encoder w/o MMD loss applied, 0-1 normalisation for each configuration, F1 = 128, F2 = 64, F3 = 32, S1=S3=1, S2 = 2 and sigmoid dense at output.

Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Ascending distance of target circuit to all circuits	1	2	3	4	5	6	7	8	9	10	11
	4*	1	5*	1*	3*	8	9*	11*	7*	24	8*
	17	4	16	17	9*	11	27	6	25	21	27
	15	17	4	16	10	27	25	27	10	20	6
	16	27	17	15	1	26	26	26	27	12	26
	2	20	9	2	20	14	11	14	26	9	14

Voltage Source | Current Source | * Circuit with difference of Resistor R2

Table 5-12 Clustering results of MMD-VAE configuration 2 @10k epochs using pair wise configuration to configuration distance.

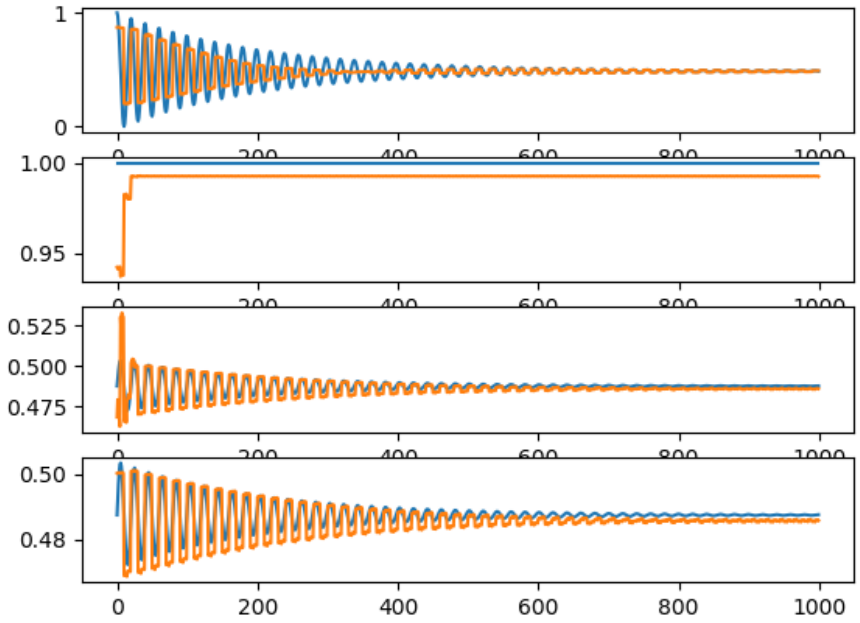


Figure 5-6 Reconstructions (in Red) of (top to bottom) Voltage and Current of R,L & C in Vs R2 RLC circuit's 1st configuration as a result of MMD-VAE configuration 2.

III. MMD-VAE configuration 3 with Z1 = 500 and Z2 =1, MMD loss applied across time dimension, 1 to -1 normalisation across all configurations of a topology, F1 = 128, F2 = 64, F3 = 32, S1=S3=1, S2 = 2 and tanh dense at output.

Target circuit indices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Ascending distance of target circuit to all circuits	1	2	3	4	5	6	7	8	9	10	11
	4*	27	5*	1*	3*	8	8	7	7*	26	8*
	17	25	16	17	1	9	11	11*	11	24	7
	15	11	4	16	10	7	9*	9	8	20	9
	21	9	17	15	20	11	2	2	27	25	2
	16	7	1	21	21	25	27	27	25	27	27

Voltage Source | Current Source | * Circuit with difference of Resistor R2

Table 5-13 Clustering results of MMD-VAE configuration 3 @10k epochs using pair wise configuration to configuration distance.

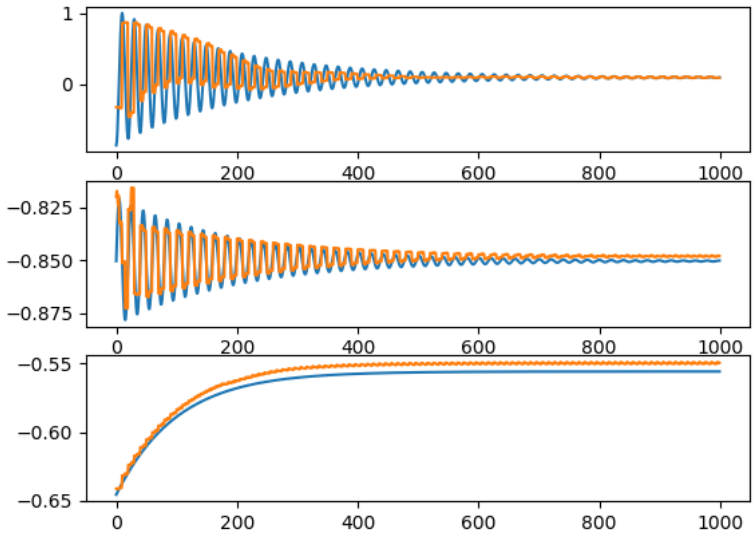


Figure 5-7 Reconstructions (in Red) of (top to bottom) Inductor Voltage and Current of Vs series RLC and Inductor Current of Is parallel RLC circuit's 1st configuration as a result of MMD-VAE configuration 3

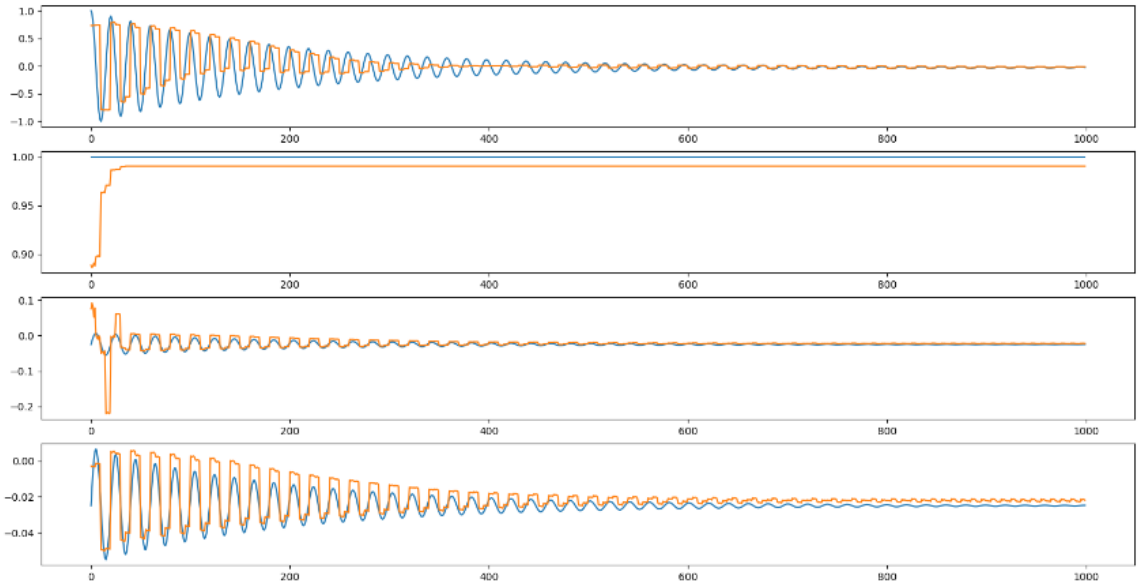


Figure 5-8 Reconstructions (in Red) of (top to bottom) Voltage and Current of R,L & C in Vs R2 RLC circuit's 1st configuration as a result of MMD-VAE configuration 3.

IV. MMD-VAE configuration 3 with Z1 = 250 and Z2 =2, MMD loss applied across signal dimension for each time step, 1 to -1 normalisation across each configurations of a topology, F1 = 128, F2 = 64, F3 = 32, S1=S2=2, S3 = 1 and tanh dense at output.

Target circuit indices	1	2	3	4	5	6	7	8	9	10	11
Ascending distance of target circuit to all circuits	1	2	3	4	5	6	7	8	9	10	11
	4*	27	5*	1*	3*	8	11	11*	7*	13	7
	3	25	16	3	16	7	9*	7	11	20	8*
	16	26	22	16	22	9	8	6	27	5	9
	5	24	4	5	13	11	6	9	2	16	2
	13	10	13	13	4	25	2	2	25	3	27

Voltage Source | Current Source | * Circuit with difference of Resistor R2

Table 5-14 Clustering results of MMD-VAE configuration 4 @2500 epochs using pair wise configuration to configuration distance.

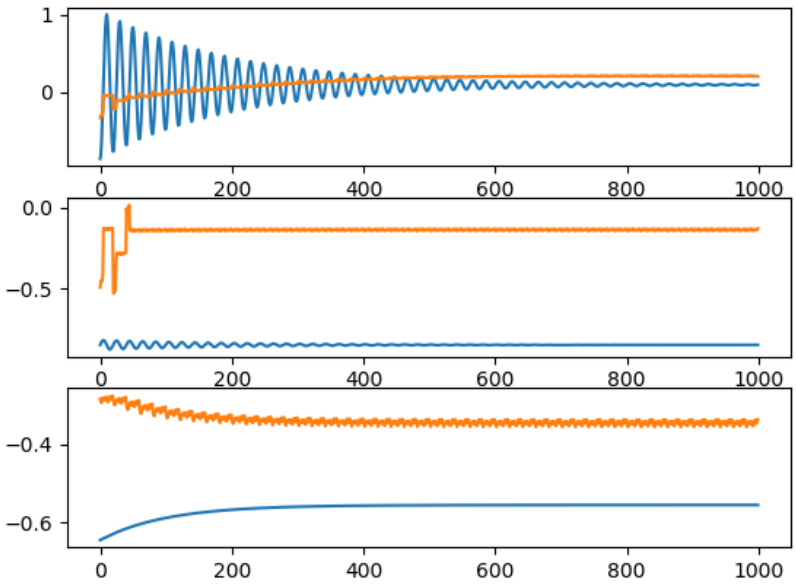


Figure 5-9 Reconstructions (in Red) of (top to bottom) Inductor Voltage and Current of Vs series RLC and Inductor Current of Is parallel RLC circuit's 1st configuration as a result of MMD-VAE configuration 4.

Ordered Circuit Configurations w.r.t C1V1	**L**	Ordered Circuit Configurations w.r.t C1V1	**RC**
C1V1	1	C1V1	2
C1V2	13	C4V1	218
C1V3	25	C1V2	14
C1V4	37	C4V2	230

C1V5	49	C1V3	26
C1V6	61	C4V3	242
C4V1	217	C1V4	38
C4V2	229	C4V4	254
C4V3	241	C1V5	50
C4V4	253	C4V5	266
C4V5	265	C1V6	62
C4V6	277	C4V6	278

Table 5-15 Row to row distance of MMD configuration 4 based on 12 rows (1 + 12x L rows and Row 2 + 12x RC rows) of each configuration (V1 - V6) of circuit Vs R2 L RC (C1) and circuit Vs L RC (C4) with "x" multiple between (1,6)

It can be seen from results in MMD VAE configurations 1 and 2 which uses normalisation across each configuration of a topology that topologies with a difference of only 1 resistor are nearest to each other whereas in MME VAE configurations 3 and 4 topologies with only 1 resistor difference are not nearer to each other but topologies under same source are mostly nearest to each other (but not perfectly for voltage source – yellow boxes are not all next to each other). Table 5-11 and Table 5-15, shows distance between rows of different Vs R2 L RC configurations (V1-V6) as well as between rows of different topologies i.e. Vs R2 L RC and Vs L RC. As Inductor path are stacked in 1st row and RC paths are stacked in 2nd row of each data input and we have used 12 rows to represent one configuration of a circuit, therefore for Vs R2 L RC inductor paths in different configurations are 1, 13, 25… and as all circuit's data is stacked across rows therefore inductor paths of Vs L RC (with circuit index number 4) are in rows 3*6*12+1, +13, +25…..Configuration 1 to 6 also represent value of resistor R from 1 to 37 Ω in intervals of 6. We can see that paths (or rows) of different configurations of both topologies are ordered w.r.t distance from path in 1st configuration based on resistor values.

It would be better for future work to use k-mean or other nearest neighbour methods to determine clusters of MMD-VAE embeddings instead of using MSE distance measure

Numerous number of experiments were conducted with different number of layers, hidden dimensions, strides, number of filters, filter size and different type of activations while keeping the main architecture same (i.e. 5 × causal 1D convolutions block shown in Figure 5-4 was always used having 1 to 3 layers), batch size = 72 and learning rate fixed at 1e-4. The clustering was mainly seems to be affected by choice of overall architecture and size of embedding dimension used instead on hyper parameters like number of layers, number of kernels etc. The clustering is also observed to be affected by choice of normalisation i.e. if row wise or circuit wise normalisation is used. It has been observed that keeping higher embedding dimension gives lower MMD loss (i.e. with 500 dimensions minimum loss across signal dimension was ≈ 0.004 and with 1000 dimension loss was ≈ 0.002).

5.5. Discussion

First assumption (stated in start of this chapter) is satisfied as all the configurations of each circuit are closer to each other than to configurations of other circuits. Whereas same topologies under different sources are not closer to each other. However different topologies with same source are closer to each other apart from some exceptions e.g. in Table 5-14 circuit number 4 is closer to 1 and then to 3 i.e. only other circuit in between but other way around there are circuits 5, 16 and 22 in between 3 and 4. These exceptions could be because of the shape (curved and/or twisted instead of flat Euclidean plane) of low dimensional manifold [78]. By looking at inductor current signals of Vs R2 C RL circuit at 2 different values of resistor R1 in Figure 5-10 and Figure 5-11, it can be said that the embedding of signal groups are based on high level features of individual signals in signal group because the general shape of signals stays same i.e. increasing (as only their rise time or time constant and steady state amplitude like characteristics of signals are affected) when resistor values are changed. Similarly in case of capacitor voltage signals of Is R2 L RC in Figure 5-12 and Figure 5-13, general signals shape is decreasing although local features are completely different. Therefore it can be said that embeddings encode general trend of each signal in a circuit across all configurations. Configurations of a topology are nearest together because all of its signals are changing in a trend (across all of its configurations) which is different from trend in signals from other topologies.

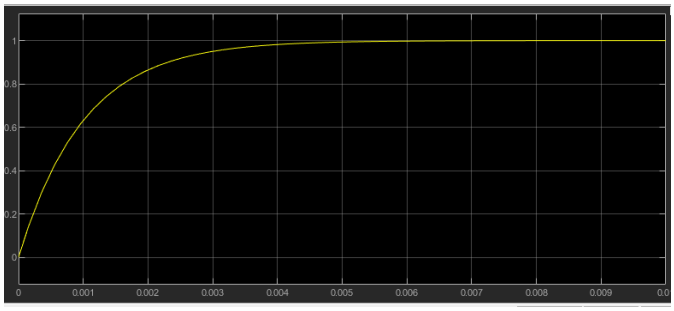


Figure 5-10 Current through inductor at R=1 Ohm in Vs R2 C RL's configuration 1

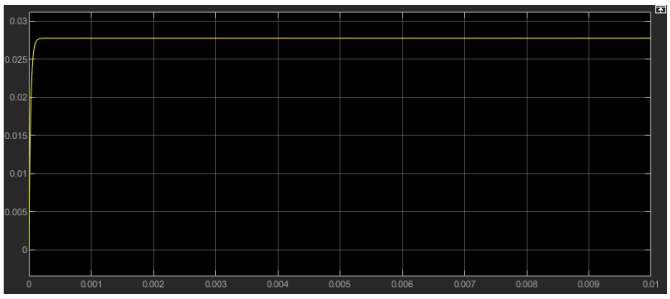


Figure 5-11 Current through inductor at R=36 Ohm in Vs R2 C RL's configuration 6

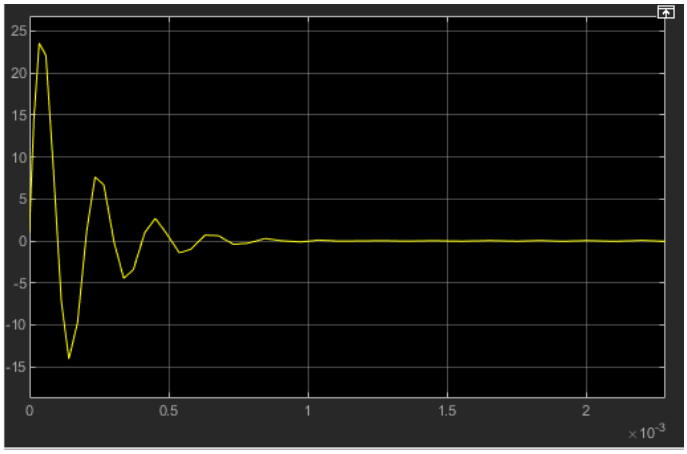


Figure 5-12 Absolute voltage on positive side of capacitor in Is R L RC's configuration 1 of R = 1 ohm

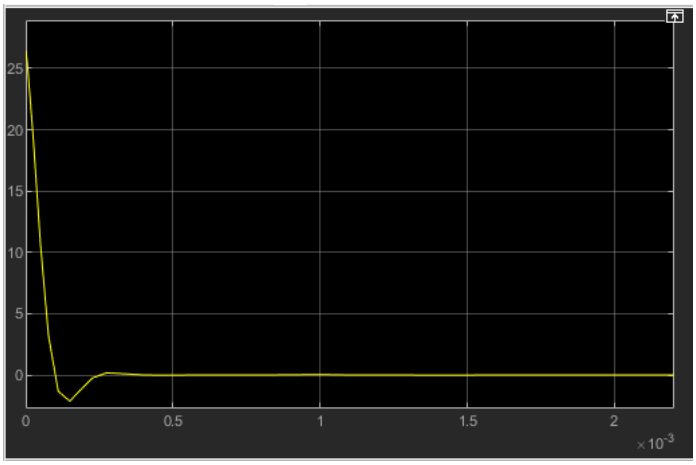


Figure 5-13 Absolute voltage on positive side of capacitor in Is R L RC's configuration 6 of R = 36 ohm

The second assumption is partially or arguably satisfied based on the test data that was created in which only one of the non-dominant component (R2) in same circuit topologies was omitted as it is difficult to ascertain which component or group of components characterises the circuit behaviour. However, a reasonable conjecture can be made i.e. changing value of dominant value will change direction of trend when non-dominant component value is changed i.e. instead signal becoming more oscillatory when non-dominant component value is changed it becomes less oscillatory. From the results, it can be seen that circuits with only a difference in presence or absence of parallel resistor R2 and having same source types are close to each other (e.g. 5 - Vs R2 C RL and 3- Vs C RL, 11 - Is R2 L RC and 8 - Is L RC).

5.6. Conclusion

The approach presented using MMD-VAE to cluster similar circuits cannot be guaranteed to generalise over any space of circuits (as we have only considered 1-11 circuits and their configurations in terms of different resistor values) or designs in different domains (e.g. mechanical) in general. Any approach to cluster similar designs can be only be guaranteed to

apply across all domains if its invariant to shift and scaling across time and amplitude but this is not needed as different MMD-VAE can be trained for different domains categorised using domain bandwidth and amplitude value ranges.

MMD VAE identifies parts belonging to a particular design based on the input and topology similarity. The paths belonging to a particular circuit (i.e. topology with particular source type) and its different configurations are grouped based on correlation between signals in all those paths. It was also noted in test results that different configurations of each circuit are ordered based on increasing resistor value (but cannot be proven to generalise however can be improved upon). This implies given a path(s) of unknown or test design having required I/O signals under required SAD (in terms of source type, input and output value range) then that unknown path(s) can be matched with path(s) from other known designs having same I/O signal characteristics. Then the unknown design can be completed by using topology and property values of those nearest known designs. The system design constraints (e.g. max input to output resistance value etc.) can be satisfied by using interpolated property values (instead of using only nearest design's property value) from multiple nearest designs. We leave this proposition to be tested in future work.

Main points to be noted are:

- Only paths from same circuit and same configuration are nearer to each other.
- Configurations of each design are ordered as per change in particular behaviour feature(s) w.r.t particular property change (e.g. increase in resistance decreases time constant of current in parallel branches as in Figure 5-10 and Figure 5-11)
- If a topology's particular configurations doesn't match its other configurations then it implies changing its properties has produced very distinct behaviour phenomenon and therefore transition to different behaviour regime or state (e.g. change in dominant component value causing signal(s) value to decrease if it was increasing before etc.).

The work conducted has only experimented by using input data format having same flow path but to understand better which type of input data format and variational auto-encoder combination leads to which type of clustering it will be interesting to use other input data formats e.g. representing paths with same effort in each row, alternatively stacking flow and effort signals in rows representing same flow path thereby increasing the dimension of input etc.

Ideally, if there exist an algorithm which can learn to relate embedding dimensions (each of which represents change in particular behaviour feature) to different combination of property values (e.g. R = 1 – 10 Ω, L = 1e-1 – 2e-1 H), or essentially to different known designs in terms

of most and least likely prospective design capable of producing the required behaviour. Then this can help to retrieve designs which has required behaviour features of output or load signals (given in requirements) and with required property values just by navigating the embedding space.

5.7. Bridging Data based Design Retrieval with Logic based Descriptive Model Retrieval

The behaviour logical architecture consists of low level functions realisable by designs similar to known designs. These low level functions (as described in section 3.4.) or operations pertaining to functions (as described in section 4.1.1.) are defined using enumerated behaviour of their parameters in form of mathematical sets (e.g. linearly increasing or decreasing over time). These sets can capture the required ideal input and output behaviour for finite values of input parameters. The initial expectation for design retrieval is that designs can be retrieved based only on the features of parameter behaviour irrespective of the scale and offset of both the required mathematical set and of simulated behaviour of known designs. The embedding space of MMD-VAE enable matching or clustering of signals based on general shape as discussed previously. The other criteria to retrieve a design using embedding is that designs represented by embedding should correspond to descriptive designs retrieved using logical or constraint satisfaction as in section 4.2. , e.g. embedded design should have at least two components with same parameter type as control and controlled parameter type.

But it is usually the case that either effort or flow signal of input and output are given in low level function e.g. “Electrical Mass Lifting” function of household domain can be given input parameter of voltage enumerated as sine wave between +/- 330 V values and output parameter of vertical velocity between -0.25 to +0.25 m/s (which can be estimated by vertical velocity values in that SAD) for both direction but without input current and output force values. Therefore, missing parameter values can be filled using commonly or most frequently occurring signal in known designs of that system application domain e.g. from known lifting system designs in household domain (w.r.t force and electricity characteristics) mean signal of input current signals corresponding to most frequent occurring load current pattern from all the designs can be used.

Chapter 6 Discussion and Future Work

6.1. Knowledge Contributions Overview

- The problem formulation presented in Chapter 3 has utilised the key concepts common across MBSE methodologies while extending some of the concepts (e.g. using abstract mathematical constraints for defining expected function behaviour) and making relations between them explicit to represent the outputs of different modelling stages in terms of models. Models utilising the formalised concepts are amenable to representation and reasoning using OWL and/or First Order Logic. This formulation has also created links between different model types by constraining parameter and property characteristics using SAD and its specialisations (which can be applied using deep learning classification of descriptive and simulation models as discussed in section 3.8.).
- Linking operational requirements to logical architecture to simulation models (representing candidate designs) by using interconnections between parameters and their characteristics has enabled checking if those requirements can be satisfied by existing designs through the novel use of neural network embedding.
- A novel framework is developed that solves the problem of knowledge organisation and retrieval for knowledge represented in different forms using logical reasoning, constraint satisfaction and neural network inference methods to aid in system model development process by relating high level system requirements to low level physical structural and behaviour models representing physically and technologically realisable components.

6.2. Knowledge Contribution based on Set Objectives

1) Investigate method of organising and reusing knowledge related to system's structure, function and behaviour for performing automatic descriptive model elaboration through model element inference as well as element to element relation inference.

- Chapter 4 has presented meta-models which links different types of models and their elements at different levels of abstractions (Engineering domain, system application domain etc.) therefore enabling use of reasoning associated to knowledge representation chosen (subsumption reasoning using OWL) as well as enabling writing custom rules using generic system-domain and specialised (as in section 4.2.1.) concepts and relations for existing model retrieval. On the other hand, references to tools are also given in which these meta-models can be implemented and refined for required purpose. Section 4.2.2. has demonstrated use of Alloy Analyser to represent

higher order predicates along with complex numerical constraints, therefore justifying practicality of set based problem formulation of Chapter 3 as well as use of concepts such as “operations” (represented by enumerated numerical tuples) in meta-models.

2) Investigate use of AI technologies to determine whether or not system with specified input to output response under given operational requirements can be realized under known technological and physical constraints.

- Section 4.2.2. and 5.4. has demonstrated use of two different type of technologies which can be used for deducing designs with logical deduction under known physical constraints (expressed in logic) and inferring designs using auto-encoder where designs are represented as solutions of complex non-linear technological (represented by simulation models) and physical constraints inside simulation environment. However in both cases, usage of required I/O behavior given in operational requirements was not demonstrated (as it requires usage of multiple tools as discussed above) although framework and processes required to achieve this is proposed in section 3.7.

3) Investigate methods to discriminate between different design configurations with different specialisations (e.g. configuration with least power consumption or high power output) based on imposed design constraints over system properties.

- Concept of system application domains and their hierarchy, w.r.t different property and parameter characteristics, serves purpose of discriminating between usage of existing design models and their elements for different application domains. The system design constraints given in requirements can be represented by specialising SADs and linking system model to that SAD. However, concrete implementation of this concept is not given as it can be tailored w.r.t available model repository and their specialisations according to use cases (e.g. car industry can configure car models for off-road, urban efficiency purposes etc.). Discriminating designs within a particular SAD based on comparing a particular property implying a particular specialisation has been discussed in section 3.8. and exemplified in section 5.5. with ordering of electronic circuits which can be linked to criteria such as power efficiency.

The declaratively encoding of knowledge has limits in terms of representing non-linear constraints on numerical values whereas to train Neural Network to learn a specific relationship requires extensive trial and error with various architectures, hyper parameter configurations and

data pre-processing methods. However Neural Networks has the potential to generalize using actual design data whereas the declarative knowledge is encoded manually by abstracting parameter space to symbolic values which reduces the design space significantly with risk of eliminating correct design solutions.

6.3. Scalability of Proposed Framework

Many of the processes shown in proposed framework (in Figure 3-6) are associated to usage of logic formalisation and constraint satisfaction, such as function allocation to physical subsystems using constraints satisfaction, representation of descriptive models using OWL/RDF knowledge bases etc. These processes has been demonstrated to scale to various use cases by previous works discussed in section 2.2. and 3.7.2.

Domain classification task or classification of designs to different SADs, albeit not implemented in this research, is a task similar to classifying interdependent multivariate temporal signals (as designs are represented using their behaviour signals). A similar and more common task involving interdependent multivariate time signals occurs in classification of human action involving 3d coordinate signals from various joints which has been demonstrated in works such as [79] on real world data.

Scalability of embedding approach is justifiable because it relies on effort and flow data, concepts borrowed from Bond Graph methodology. This methodology is used in modelling continuous and discrete behaviour of mechatronic systems from disparate domains. Every respectable engineering organisation nowadays have a repository of mechatronic simulation models capturing various aspects of their products and configured to various potential use cases which can be encountered by their customer as well as configured to variations of their products.

The knowledge based model analysis process in framework of Figure 3-6 only acts as an deterministic decision making rule based interface between input/query requirement, descriptive design retrieval and machine learning algorithm, which can be implemented with either declarative or procedural programming hence can be scaled by writing rules based on complexity of the application domain (e.g. to compare the retrieved and classified design from NN embedding task with respect to global property values given in requirement model).

Proposed architecture cannot be implemented in a context of engineering business as it is because overall framework still require further validation which includes conversion of the outputs of logic reasoning tasks to input of machine learning tasks and implementation of SAD classification framework for both descriptive and embedding representation of physical designs to test system design constraints.

6.4. Future Work

This section consolidates future works proposed throughout the chapter.

- System modelling concepts given in Chapter 3 are not fully mathematically defined (as only examples are used for defining requirements and functions) and therefore can be improved further by learning about usage of MBSE modelling concepts in particular application domain with matured model repositories (e.g. Automobile) for creating definitions expressive enough to cover all modelling use cases (e.g. use of functions in failure mode effect analysis).
- A proposal is made in section 3.7. of using deep learning classifier(s) for classifying existing system physical (or simulation) models to different SAD specialisations pertaining to use cases of a particular industry whereas descriptive models can be classified by manually building descriptive knowledge base of SAD ontologies and then using rules to specify criteria of classifications (Although graph based machine learning techniques can also be used e.g. by combining techniques given in [80] and [81]).
- Design constraint violation inspection using SAD classification mechanism can also be implemented to test if the retrieved full system design (incorporating integrated sub-system designs) satisfies system design property or global property constraints by checking if it can be classified[36] to SAD given in the system requirement model.
- The model analysis task discussed in section 3.7.3. interfaces/consolidates different streams of information which can be implemented as data fusion process by utilising either logical or machine learning framework or combination of both (with machine learning at low level providing probable predictions to logic) for particular subtask arising from the two main knowledge based model analysis tasks (given at start of section 3.7.) e.g. machine learning algorithm can be given following inputs to rank retrieved designs (pertaining to same SAD): the relations between descriptive models elements, decomposition of particular function and its interaction with other functions using standardised flows [65].. However this will require converting symbolic information to numerical information by using different adjacency matrix for each relation type and by using columns of a matrix to represent different attributes of a model element.

[36] Multiple chained classifiers may be needed e.g. first classifier can be trained to classify subsystems with multiple types of property values (can be one or more than of same type) belonging to different SADs and then another classifier can be trained to give score for classifications belonging to each subsystem group forming one design

- Implementation of framework demonstrating inference of physical designs with auto-encoder by using input of required I/O behaviour given in requirement/logical architecture model and using descriptive designs inferred by applying logical deduction.
- Cross domain application of NN embedding approach to check scalability across systems of varying time constants as well as across systems types with hybrid (discrete and continuous) behaviour.
- Comparison between clustering performance of NN embedding approach with different input formats having different arrangement of effort and flow signals e.g. instead of concatenating effort signals from same flow path in a row as it is done in Chapter 5, flow signals from branches with same effort value can be concatenated.
- Experiments with MMD-VAE in section 5.4. used configuration to configuration distance to order circuits to represent clusters however better generalisation can be demonstrated if row to row distance (i.e. distance between embedding of branches or parts of circuits) is used to order circuits to represent clusters and if similar or better clustering performance can be achieved. This would mean that circuits or designs with matching parts can be retrieved.
- Instead of using MSE distance measure in determining clusters in MMD-VAE experiments of section 5.4. , K-mean or other nearest neighbour methods which are more suited to non-Euclidean nature of the embedding space can be used.
- Specialising clustering algorithm by supervised training to target cluster with particular characteristic e.g. to cluster circuits having similar behaviour over one of their component for same inputs (underdamped, critically or overdamped behaviour) or to cluster circuits having same effective/net property value as seen by a particular component (e.g. as seen by load resistor of particular value common across circuits).
- Clustering of designs or circuits using behaviour data generated with input profiles common in use cases of a particular SAD.

Chapter 7 References

[1] S. Friedenthal, A. Moore and R. Steiner, "Systems Engineering Overview," in *A Practical Guide to SysML,* Waltham, MA (USA), Elsevier Inc, 2012, pp. 12-13.

[2] A. Goknil, I. Kurtev and J. Millo, "A Metamodeling Approach for Reasoning on Multiple Requirements Models," in *17th IEEE International Enterprise Distributed Object Computing Conference*, Vancouver, BC, 2013.

[3] S. Friedenthal, A. Moore and R. Steiner, A Practical Guide to SysML, Waltham, MA (USA): Elsevier Inc, 2012.

[4] J. Johnson, "Model Based Systems Engineering Wiki," INCOSE UK, 2013. [Online]. Available: http://www.incosewiki.info/Documents/Site_Resources/Files/MBSE/JJ_Analysis_of_INCOSE_UK_MBSE_WG_2013_Survey_v2.pdf. [Accessed 27 June 2016].

[5] M. Blackburn, R. Cloutier, G. Witus, E. Hole and M. Bone, "Transforming System Engineering through Model centric engineering," Stevens Institute of Technology, Jersy, USA, 2015.

[6] P. Roques, "MBSE with the ARCADIA Method and the Capella," in *8th European Congress on Embedded Real Time Software and Systems*, Toulouse,, 2016.

[7] A. M. Prado, "Leveraging Ontology Technologies for Data Modeling in Space Engineering (Master Thesis)," Luleå University of Technology, Luleå (Sweden), 2013.

[8] M. Blackburn and P. Denno, "Using Semantic Web Technologies for Integrating Domain Specific Modeling and Analytical Tools," *Procedia Computer Science, Conference on Complex Adaptive Systems,* vol. 61, pp. 141-146, 2015.

[9] H. J. Lee, S. Fenves, C. Bock, W. H. Suh, S. Rachuri, X. Fiorentini and R. Sriram, "A Semantic Product Modeling Framework and Language for behavior

evaluation," Manufacturing Systems Integration Division Manufacturing Engineering Laboratory, National Institute of Standards and Technology, USA, 2009.

[10] S. Ponnusamy, P. Thebault and V. Albert, "Towards an ontology-driven framework for simulation model development," *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016),* p. 11, 2016.

[11] M. Cencetti, "Evolution of Model-Based System Engineering Methodologies for the Design of Space Systems in the Advanced Stages of the Project (Phases B-C), Doctoral thesis," Politecnico di Torino, Torino (Italy), 2014.

[12] J. McGuire, D. Kuokka, J. Weber, J. Tenenbaum, T. Gruber and G. Olsen, "SHADE: Technology for Knowledge-based Collaborative Engineering," *Concurrent Engineering: Research and Applications,* vol. 1, no. 3, pp. 137-146, 1993.

[13] X. Zha, R. Sriram, M. Fernandez and F. Mistree, "Knowledge-intensive collaborative decision support for design processes: A hybrid decision support model and agent," *Computers in Industry,* vol. 59, no. 805-922, p. 9, 2008.

[14] M. Blackburn and P. Denno, "Virtual Design and Verification of Cyber-physical Systems: Industrial Process Plant Design," *Procedia Computer Science, 2014 Conference on Systems Engineering Research,* vol. 28, pp. 883-890, 2014.

[15] M. Campbell, J. Cagan and K. Kotovsky, "A-Design: An Agent-Based Approach to Conceptual Design in a Dynamic Environment," *Research in Engineering Design,* vol. 11, no. 3, pp. 172-192, 1999.

[16] A. Chakrabarti, K. Shea, R. Stone, J. Cagan, M. Campbell, V. N. Hernandez and K. Wood, "Computer-Based Design Synthesis Research: Overview," *Journal of Computing and Information Science in Engineering,* vol. 11, no. 2, p. 10, 2011.

[17] A. Levy, Y. Iwasaki and R. Fikes, "Automated model selection for simulation based on relevance reasoning," *Artificial Intelligence,* vol. 96, no. 2, pp. 351-394, 1997.

[18] C. J. Bejarano, T. Coudert, E. Vareilles, L. Geneste and J. Abeille, “Case-based reasoning and system design: An integrated approach based on ontology and preference modeling,” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, no. 1, pp. 49-69, 2014.

[19] S. Marcus, J. Stout and J. McDermott, “VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking,” *AI Magazine*, vol. 1, no. 9, p. 95–112, Spring 1988.

[20] N. Nassara and M. Austin, “Model-Based Systems Engineering Design and Trade-Off Analysis with RDF Graphs,” *Procedia Computer Science, 2013 Conference on Systems Engineering Research*, vol. 16, pp. 216-225, 2013.

[21] F. X. Zha, “A knowledge intensive multi-agent framework for cooperative/collaborative design modeling and decision support of assemblies,” *Knowledge-Based Systems*, vol. 15, pp. 493-506, 2002.

[22] S. Jenkins and N. F. Rouquette, “Semantically-Rigorous Systems Engineering Modeling Using Sysml and OWL,” in *nternational Workshop on Systems & Concurrent Engineering for Space Applications*, Lisbon, Portugal, 2012.

[23] H. Mehrpouyan, I. Tumer, C. Hoyle, D. Giannakopoulou and G. Brat, “Formal Verification of Complex Systems based on SysML Functional Requirements,” in *ANNUAL CONFERENCE OF THE PROGNOSTICS AND HEALTH MANAGEMENT SOCIETY*, Fort Worth, Texas, 2014.

[24] R. Kalawsky, J. Brien, S. Chong, C. Wong, J. Jia, H. Pan and P. Moore, “Bridging the Gaps in a Model Based System Engineering Workflow by Encompassing Hardware-in-Loop Simulation,” *IEEE Systems Journal*, vol. 7, no. 4, pp. 593-605, 2013.

[25] S. Wölkl J, “Model Libraries for Conceptual Design (Phd Thesis),” Technical University of Munich, Munich, 2013.

[26] B. Kruse and K. Shea, “Design Library Solution Patterns in SysML for concept desigjn and simulation,” in *26th CIRP Design Conference*, Stockholm, Sweden, 2016.

[27] PolarSys, "Validation Rules," 2013. [Online]. Available: http://polarsys.org/help/index.jsp?topic=%2Forg.polarsys.capella.validation.doc%2Fhtml%2FValidation+Rules%2Fintegrity%2FValidationRules.html. [Accessed 22 10 2016].

[28] M. Campbell, B. Fernandez and Z. Wu, "Bond Graph Based Automated Modelling for Computer-Aided Design of Dynamic Systems," *Journal of Mechanical Design,* vol. 130, no. 4, p. 11, 2008.

[29] M. Campbell, J. Cagan and K. Kotovsky, "Agent-Based Synthesis of Electromechanical Design Configurations," *ASME Journal of mechanical design,* vol. 122, no. 1, pp. 61-69, 2000.

[30] K. J. Gui and M. Mäntylä, "New concepts for complete product assembly modeling," *Proceedings on the second ACM symposium on Solid modeling and applications,* pp. 397-406, 1993.

[31] B. Smyth, D. Finn and M. Keane, "Design Synthesis: A Model of Hierarchical Case-Based Reasoning," Trinity College Dublin, Dublin, 1993.

[32] A. Morkevicius and N. Jankevicius, "An approach: SysML-based automated requirements verification," in *2015 IEEE International Symposium on Systems Engineering (ISSE)*, Rome, 2015.

[33] J. Estefan, "Survey of Model-Based Systems Engineering (MBSE) Methodologies," International Council on Systems Engineering (INCOSE), 23 05 2008. [Online]. Available: www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf. [Accessed 11 07 2016].

[34] S. K, T. E, Z. Y, P. J and A. U, "Towards Ontology-driven Requirements Engineering," in *The 10th International Semantic Web Conference*, Bonn, Germany, 2011.

[35] A. Goknil, I. Kurtev, K. van den Berg and J. and Veldhuis, "Semantics of trace relations in requirements models for consistency checking and inferencing," *Software & Systems Modeling,* vol. 10, no. 1, pp. 31-54, 2011.

[36] H. Graves, “Integrating SysML and OWL,” *Proceedings of the 6th International Conference on OWL: Experiences and Directions,* vol. 529, pp. 117-124, 2009.

[37] D. Wagner, M. Bennett, R. Karban, N. Rouquette, S. Jenkins and M. and Ingham, “An ontology for State Analysis: Formalizing the mapping to SysML,” in *2012 IEEE Aerospace Conference,*, Big Sky, MT, 2012.

[38] S. Feldmann, K. Kernschmidt and B. Vogel-Heuser, “Combining a SysML-based modeling approach and semantic technologies for analyzing change influences in manufacturing plant models,” *Variety Management in Manufacturing: Proceedings of the 47th CIRP Conference on Manufacturing System,* p. 451 – 456, 2014.

[39] C. MÜNZER HEINZ, “Constraint-Based Methods for Automated Computational - Doctoral Thesis,” ETH ZURICH, ZURICH, 2015.

[40] B. Helms, “Object-Oriented Graph Grammars for computational design synthesis,” Technical university of Munich, Munich, Germany, 2012.

[41] S. Friedenthal, A. Moore and R. Steiner, “Modelling principles,” in *A Practical Guide to SysML,* Waltham, MA, USA, Elsevier, 2012, p. 21.

[42] L. Baker, P. Clemente, B. Cohen, L. Permenter, B. Purves and P. Salmon, “Foundational Concepts for Model Driven System Design (white paper),” INCOSE Model Driven System Design Interest Group, International Council on Systems Engineering, 2000.

[43] P. H. Hoffman, “Deskbook: Model Based System Engineering with Rational Rhapsody and Rational Harmony for System Engineering,” IBM Corporation, Somers, 2011.

[44] D. F. Giorgio, V. Paparo, V. Basso and X. Roser, “Applying Collaborative System Engineering in Thales Alenia Space: lessons learned and best practices,” Thales Alenia Space, Italy, 2012.

[45] T. Berners-Lee, J. Hendler and O. and Lassa, “The Semantic Web,” *Scientific American,* pp. 35-43, 2001.

[46] K. Siegemund, E. Thomas, Y. Zhao, J. Pan and U. Assmann, “Towards Ontology-driven Requirements Engineering,” in *The 10th International Semantic Web Conference*, Bonn, Germany, 2011.

[47] H. Graves, “Integrating Reasoning with SysML,” *INCOSE International Symposium*, vol. 22, no. 1, pp. 2228-2242, 2014.

[48] M. Austin and N. Nassar, “Model-Based Systems Engineering Design and Trade-Off Analysis with RDF Graphs,” *Procedia Computer Science, 2013 Conference on Systems Engineering Research*, vol. 16, pp. 216-225, 2013.

[49] R. Hodgson, “NExIOM, the NASA Constellation Program Ontologies,” in *11th NASA-ESA Workshop on Product Data Exchange*, Kent Space Center, 2009.

[50] F. J. Castet, M. Rozek, M. Ingham, N. Rouquette, S. Chung, S. Jenkins, D. Wagner and D. Dvorak, “Ontology and Modeling Patterns for State-Based Behavior Representation,” in *AIAA Infotech @ Aerospace, AIAA SciTech Forum*, Kissimmee, Florida, 2015.

[51] J. Dong, Y. Zhao and T. Peng, “A Review of Design Pattern Mining Techniques,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 9, no. 6, pp. 823-852, 2009.

[52] K. R. Keller, R. Schauer, S. Robitaille and P. Page, “Pattern-Based Reverse-Engineering of Design Components,” in *Proceedings of the 21st international conference on Software engineering*, Los Angeles, California, USA, 1999.

[53] S. PONNUSAMY, V. ALBERT and P. THEBAULT, “A meta-model for automatic simulation model selection,” in *uropean Simulation and Modeling Conference (ESM)* , Leicester, 2015.

[54] P. S, P. Thebault and V. Albert, “Towards an ontology-driven framework for simulation model development,” *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, p. 11, 2016.

[55] J. Lin, M. Fox and T. Bilgic, “A Requirement Ontology for Engineering Design (1996),” *Concurrent Engineering*, vol. 4, no. 4, pp. 279-291, 1996.

[56] W. Schindel, “Requirements statements are transfer functions: An insight from model-based systems engineering,” in *Proceedings of INCOSE 2005 International Symposium*, Las Vegas, 2005.

[57] W. Schindel, “Pattern-Based Systems Engineering (PBSE), Based On S*MBSE Models,” INCOSE MBSE Patterns Working Group, Online, 2015.

[58] S. Xia, D. Linkens and S. Bennett, “Automatic modelling and analysis of dynamic physical systems using qualitative reaasoning and bond graphs,” *Intelligent Systems Engineering*, vol. 2, no. 3, pp. 201-212, 1993.

[59] N. Ivezica and J. Garrett, “A neural network-based machine learning approach for supporting synthesis,” *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, vol. 8, no. 2, pp. 143-161, 1994.

[60] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, http://www.deeplearningbook.org: MIT Press, 2016.

[61] I. Goodfellow, Y. Bengio and A. Courville, “Principal Components Analysis,” in *Deep Learning*, http://www.deeplearningbook.org, MIT Press, 2016, pp. 48-52.

[62] I. Goodfellow, Y. Bengio and A. Courville, “Neural Language Models,” in *Deep Learning*, http://www.deeplearningbook.org, MIT Press, 2016, pp. 464-465.

[63] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” in *The International Conference on Learning Representations (ICLR)*, Banff, 2014.

[64] S. Zhao, J. Song and S. Ermon, “InfoVAE: Balancing Learning and Inference in Variational Autoencoders,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 10.1609/aaai.v33i01.33015885 , pp. 5885-5892, 2018.

[65] J. Hirtz, R. Stone, D. McAdams, S. Szykman and K. and Wood, “A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts,” National Institute of Standards and Technology, Washington, 2002.

[66] F. Baader, I. Horrocks and U. Sattler, “Description Logics,” in *Handbook of Knowledge Representation*, Oxford, Elsevier, 2008, pp. 115-180.

[67] S. Friedenthal, A. Moore and R. W. Steiner, “Applying OOSEM to Specify and Design the Residential Security System,” in *A Practical Guide to SysML*, MA (USA) , Elsevier Inc, 2012, p. 444.

[68] P. Struss, “Model-based Problem Solving,” in *Handbook of Knowledge Representation*, Oxford, Elsevier, 2008, pp. 455-456.

[69] H. Paynter, Analysis and design of engineering systems, Cambridge, MA: MIT Press, 1961.

[70] F. J. Broenink, “Introduction to Physical Systems Modeling with Bond Graphs,” in *in the SiE whitebook on Simulation Methodologies*, 1999.

[71] K. D. Forbus, “Qualitative Modelling,” in *Handbook of Knowledge Representation*, Oxford, Elsevier, 2008, pp. 361-394.

[72] I. Goodfellow, J. Abadie Pouget, M. Mirza, B. Xu, W. D. Farley, S. Ozair, A. Courville and Y. Bengio, “Generative Adversarial Networks,” in *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, Montreal, Canada, 2014.

[73] L. E. Zec, H. Arnelid and N. Mohammadiha, “Recurrent Conditional GANs for Time Series Sensor Modelling,” in *Time Series Workshop at International Conference on Machine*, Long Beach, California , 2019.

[74] S. Bai, Z. J. Kolter and V. Koltun, An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling, arXiv: 1803.01271v2, 2018.

[75] A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, S. A.W. and K. Kavukcuoglu, WaveNet: A Generative Model for Raw Audio, arXiv: 1609.03499, 2016.

[76] V. D. O. Oord, N. Kalchbrenner and K. Kavukcuoglu, “Pixel Recurrent Neural Networks,” in *Proceedings of the 33 rd International Conference on Machine*, New York, NY, USA, 2016.

[77] H. Salimbeni and P. M. Deisenroth, “Doubly Stochastic Variational Inference for Deep Gaussian Processes,” in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, California, 2017.

[78] I. Goodfellow, Y. Bengio and A. Courville, “Learning Manifolds with Autoencoders,” in *Deep Learning*, http://www.deeplearningbook.org, MIT Press, 2016, pp. 515-520.

[79] S. Agahian, F. Negin and C. Köse, “An efficient human action recognition framework with pose-based spatiotemporal features,” *Engineering Science and Technology,an International Journal*, vol. 23, no. 1, pp. 196-203, 202.

[80] P. Veličković, W. Fedus, W. Hamilton L, P. Liò, Y. Bengio and D. R. Hjelm, “Deep Graph Infomax,” in *International Conference on Learning Representations*, New Orleans, Louisiana, United States, 2019.

[81] M. Brockschmidt, M. Allamanis, A. Gaunt and O. Polozov, “GENERATIVE CODE MODELING WITH GRAPHS,” in *International Conference on Learning Representations*, New Orleans, Louisiana, United States, 2019.

[82] D. L. Davies and D. W. Bouldin, “A Cluster Separation Measure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vols. PAMI-1, no. 2, p. 224–227, 1979.

[83] A. Kristiadi, “Variational Autoencoder: Intuition and Implementation,” Github, 10 December 2016. [Online]. Available: https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/. [Accessed 5 September 2020].

[84] A. Ng, “CS294A Lecture notes Sparse autoencoder,” Stanford University, 2011. [Online]. Available: https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf. [Accessed 19 September 2020].

[85] O. M. Group, “OMG Systems Modeling Language Specification,” Object Management Group, Inc, 2015. [Online]. Available: http://www.omg.org/spec/SysML/1.4/. [Accessed 18 11 2016].

[86] R. Uppaal, B. Kucharski, P. B. Bhanu Pratap Singh, I. Deznabi and M. Fiterau, "Multi-resolution Attention with Signal Splitting for Multivariate Time Series," in *Proceedings..*, 2019.

Chapter 8 Appendix

8.1. Comparison Between Auto-encoder Methods for Electrical Circuit Clustering

Auto-encoder architecture shown in section 5.4. is used to compare performance of applying different loss functions. The DB index[37] [82] and classification accuracy measure are used as performance metrics. Accuracy measure uses classification of each circuit's embedding which is defined based on embedding's average distance to nearest cluster centroid. Distance average is calculated over all paths/branches of all configurations of a circuit. The cluster centroid is calculated by averaging over embedding of all 6 configurations of each circuits belonging to a particular class.

$A_i = \frac{1}{N}\sum_{k=1}^{N} M_i \times Z_k$, where Z is $R^{N\times LX\ H}$ and M is binary mask of $i \in$Q class of dimension $\mathbb{Z}^N$,

N is number of circuits, H is embedding dimension and L = C × P, where C is number of configurations (which are 6 for each circuit) and P is number of Paths (which are 12 for each configuration of a circuit).

Class is then assigned to a circuit which has minimum average distance to all paths and configuration of a circuit i.e. for a circuit k,

$$\mathrm{aD}_{ki} = \frac{1}{C\times P}\sum_{j=0}^{C\times P} ||\mathrm{Z}_j^k - \mathrm{A}_i||_2$$

and class is selected with $\mathrm{argmin}_{\,i\in Q}$ aD

DB index is calculated by,

$\max_{\,i,j\in Q,\, i\neq j} \frac{\frac{1}{N}\sum_{k=1}^{N} M_i\times aD_i + \frac{1}{N}\sum_{k=1}^{N} M_j\times aD_j}{\frac{1}{C\times P}\sum_{n=1}^{C\times P}\|Ai-Aj\|_2}$ for N circuits and classes i and j except i =j

Following three types of clusters (as proposed in section 5.4.) have been used to compare performance of 4 different auto-encoder methods.

8.1.1. Cluster with difference of R2 and with same source

- Class 1 = {1-Vs R2 L CR, 4-Vs L CR and 26-VsR2 L CR }
- Class 2 = {3-Vs C LR, 5-Vs R2 C LR and 24-VsR2 C LR }

[37] This index gives measure of how well separated clusters are from each other and how dense each of the cluster is.

- Class 3 = {8-Is L CR, 11-Is R2 L CR and 27-IsR2 L CR}
- Class 4 = {7-Is C LR, 9- Is R2 C LR and 25-IsR2 C LR }
- Class 5 = {12-VsR2L R C, 18-VsL C R R2 and 20-VsR2 C R L}
- Class 6 = {13-VsR2C R L, 16-VsC L R R2 and 22-VsR2 L R C }
- Class 7 = {14-IsR2L R C, 19-IsL C R R2 and 21-IsR2 C R L}
- Class 8 = {15-IsR2C R L, 17-IsC L R R2 and 23-IsR2 L R C }
- Class 9 = {6-Vs R2 RLC and 28-Vs series RLC}
- Class 10 = {10-Is R2 RLC, 2- Is parallel RLC}

1) Vanilla Autoencoder: Removing latent loss from auto encoder architecture of section 5.4.

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	15	21	16	2
2	2	27	25	11	9	28	1
3	3	16	5	4	17	1	2
4	4	1	17	16	15	21	2
5	5	3	10	1	20	21	2
6	6	8	9	7	11	25	1
7	7	8	11	9	2	27	2
8	8	7	11	9	2	27	2
9	9	11	7	8	27	25	2
10	10	26	24	20	25	27	1
11	11	8	7	9	2	27	2
12	12	21	20	15	23	1	2
13	13	20	26	24	10	25	1
14	14	2	28	25	27	11	1
15	15	21	1	4	17	12	2
16	16	17	4	1	3	15	1
17	17	4	16	1	15	21	2
18	18	19	28	2	14	21	1
19	19	18	28	4	17	21	1
20	20	10	26	24	21	13	1
21	21	15	20	12	1	23	1
22	22	23	12	20	24	14	1
23	23	21	12	15	20	1	2
24	24	26	25	27	10	20	1

25	25	27	2	24	26	9	1
26	26	24	25	27	10	20	2
27	27	25	2	9	11	26	2
28	28	2	27	25	9	11	1
Total							42

Table 8-1 Same topology (R2 difference) and same source clustering results from Vanilla Autoencder

2) Modifying auto encoder architecture from section 5.4. by adding additional dense layer at end of encoder to produce parameterised standard deviation ($\Sigma^{1/2}$) as well as adding sampling trick to sample embedding from parameterised embedding distribution. VAE loss associated to a normal distribution implementation from [83] is used.

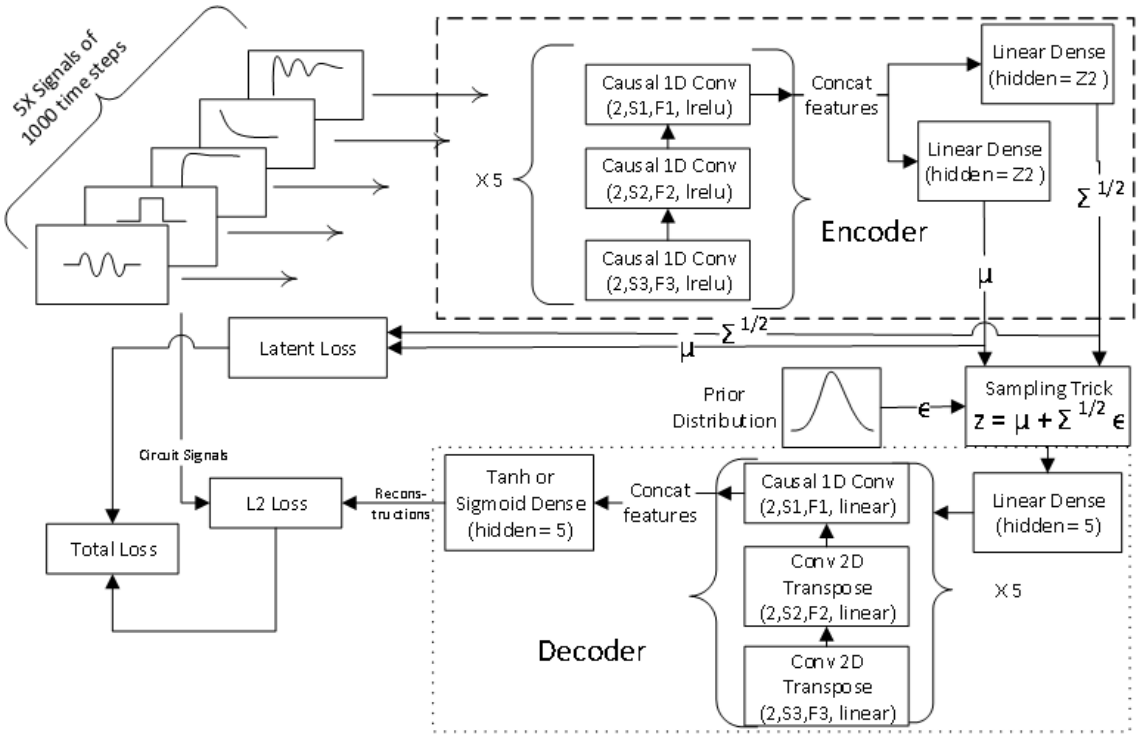


Figure 8-1 VAE architecture used for clustering similar circuits

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	15	23	21	2
2	2	11	1	17	4	15	1
3	3	5	19	4	17	1	2
4	4	1	17	15	23	21	2
5	5	3	19	1	4	17	2

6	1	4	17	15	23	21	1
7	5	3	19	17	1	4	1
8	2	11	17	1	4	15	1
9	5	3	17	1	4	15	0
10	10	17	15	4	1	23	1
11	11	2	14	1	15	17	1
12	12	4	1	17	15	23	1
13	13	17	4	1	15	23	1
14	14	12	24	4	17	15	1
15	15	4	17	1	23	21	3
16	16	17	4	1	15	23	1
17	17	4	1	15	23	21	3
18	18	19	13	4	1	17	1
19	19	4	1	17	15	23	1
20	20	19	1	4	17	15	1
21	21	1	4	17	15	23	1
22	22	4	17	1	15	23	1
23	23	15	17	4	1	21	3
24	24	17	4	1	15	23	1
25	24	15	23	1	17	4	0
26	26	17	4	1	15	13	2
27	26	15	23	1	17	4	0
28	3	5	19	21	4	1	0
Total							35

Table 8-2 Same topology (R2 difference) and same source clustering results from VAE Autoencdoer

3) Using MMD-VAE configuration 4 from section 5.4.

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	16	17	5	3	2
2	2	27	25	26	24	9	1
3	3	5	4	16	1	17	2
4	4	1	16	17	3	5	2
5	5	3	1	4	16	10	2
6	6	8	7	2	25	11	1
7	7	8	11	9	6	2	2
8	8	6	7	11	9	2	2

9	9	11	7	2	27	25	2
10	10	13	20	5	1	26	1
11	11	9	7	8	2	25	2
12	12	22	13	10	1	20	1
13	13	10	20	21	1	15	1
14	10	14	13	26	12	20	1
15	15	13	21	1	4	20	1
16	16	4	17	1	3	5	1
17	17	16	4	1	3	19	1
18	18	19	21	3	13	20	1
19	19	18	4	3	17	21	1
20	20	13	21	10	22	1	1
21	21	20	13	19	1	15	2
22	22	12	20	13	10	21	2
23	23	22	15	21	12	13	2
24	24	26	25	2	27	10	1
25	25	27	2	24	26	9	1
26	26	24	2	27	10	25	2
27	27	2	25	26	24	9	1
28	28	10	26	13	14	20	1
Total							40

Table 8-3 Same topology (R2 difference) and same source clustering results from MMD configuration 4

4) Sparse Autoencoder: Using auto encoder architecture of section 5.4. with leaky relu activations in encoder replaced by sigmoid activation and sparse regularisation applied on activations of all internal layers of encoder (except embedding layer) using implementation of [84] from https://github.com/zhiweiuu/sparse-autoencoder-tensorflow with $\beta = 0.01$ and $\rho = 0.01$.

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	16	21	15	2
2	2	27	25	26	24	28	1
3	3	16	5	17	28	4	2
4	4	1	17	16	15	21	2
5	5	3	10	26	20	21	2
6	6	8	7	11	9	25	1
7	7	8	11	6	9	2	2

8	8	7	6	11	9	25	2
9	9	11	7	8	27	25	2
10	10	26	20	21	12	24	1
11	11	7	8	9	6	2	2
12	12	21	20	10	26	24	2
13	13	15	20	21	26	12	1
14	14	2	24	25	12	23	1
15	15	13	21	20	12	1	1
16	16	17	4	1	3	15	1
17	17	16	4	1	3	15	1
18	18	19	28	2	27	3	1
19	19	18	28	2	27	3	1
20	20	21	12	26	10	15	2
21	21	20	12	10	15	26	1
22	22	23	24	12	25	26	1
23	23	22	12	20	21	24	1
24	24	26	25	27	10	20	1
25	25	27	24	26	2	10	1
26	26	27	24	25	10	20	2
27	27	25	26	24	2	10	1
28	28	2	1	4	3	26	1
Total							39

Table 8-4 Same topology (R2 difference) and same source clustering results from Sparse Autoencoder

Auto-encoder Type	Accuracy	DB index	Number of circuits in same cluster as Target circuit (out of first 5)
Vanilla	0.25	183.91	**42**
VAE	**0.75**	**52.031**	35
MMD	0.43	190.82	40
Sparse	0.11	2611.35	39

Table 8-5 Same source and similar topology clustering performance metrics from 4 different auto-encoder architectures

8.1.2. Circuits with similar topology irrespective of source type

- Class 1 = {4-Vs L CR, 8-Is L CR, 1-Vs R2 L CR, 11-Is R2 L CR, 26-VsR2 L CR and 27-IsR2 L CR}
- Class 2 = {3-Vs C LR, 7-Is C LR, 5-Vs R2 C LR, 9- Is R2 C LR, 24-VsR2 C LR and 25-IsR2 C LR}

- Class 3 = {2- Is parallel RLC}
- Class 4 = {12-VsR2L R C, 14-IsR2L R C, 18-VsL C R R2, 19-IsL C R R2, 20-VsR2 C R L and 21-IsR2 C R L }
- Class 5 = {13-VsR2C R L, 15-IsR2C R L, 16-VsC L R R2, 17-IsC L R R2, 22-VsR2 L R C and 23-IsR2 L R C }
- Class 6 = {6-Vs R2 RLC, 10-Is R2 RLC and 28-Vs series RLC}

1) Vanilla Autoencoder

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	15	21	16	2
2	2	27	25	11	9	28	1
3	3	16	5	4	17	1	2
4	4	1	17	16	15	21	2
5	5	3	10	1	20	21	2
6	6	8	9	7	11	25	1
7	7	8	11	9	2	27	2
8	8	7	11	9	2	27	2
9	9	11	7	8	27	25	2
10	10	26	24	20	25	27	1
11	11	8	7	9	2	27	2
12	12	21	20	15	23	1	3
13	13	20	26	24	10	25	1
14	14	2	28	25	27	11	1
15	15	21	1	4	17	12	2
16	16	17	4	1	3	15	2
17	17	4	16	1	15	21	2
18	18	19	28	2	14	21	3
19	19	18	28	4	17	21	2
20	20	10	26	24	21	13	2
21	21	15	20	12	1	23	3
22	22	23	12	20	24	14	2
23	23	21	12	15	20	1	2
24	24	26	25	27	10	20	2

25	25	27	2	24	26	9	2
26	26	24	25	27	10	20	2
27	27	25	2	9	11	26	2
28	28	2	27	25	9	11	1
Total							53

Table 8-6 Same topology (R2 difference) and irrespective of source clustering results from Vanilla Autoencoder

2) VAE Autoencoder

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	15	23	21	2
2	2	11	1	17	4	15	1
3	3	5	19	4	17	1	2
4	4	1	17	15	23	21	2
5	5	3	19	1	4	17	2
6	1	4	17	15	23	21	0
7	5	3	19	17	1	4	2
8	2	11	17	1	4	15	2
9	5	3	17	1	4	15	2
10	10	17	15	4	1	23	1
11	11	2	14	1	15	17	2
12	12	4	1	17	15	23	1
13	13	17	4	1	15	23	3
14	14	12	24	4	17	15	2
15	15	4	17	1	23	21	3
16	16	17	4	1	15	23	3
17	17	4	1	15	23	21	3
18	18	19	13	4	1	17	2
19	19	4	1	17	15	23	1
20	20	19	1	4	17	15	2
21	21	1	4	17	15	23	1
22	22	4	17	1	15	23	3
23	23	15	17	4	1	21	3
24	24	17	4	1	15	23	1
25	24	15	23	1	17	4	1
26	26	17	4	1	15	13	2
27	26	15	23	1	17	4	2

28	3	5	19	21	4	1	0
Total							51

Table 8-7 Same topology (R2 difference) and irrespective of source clustering results from VAE Autoencoder

3) MMD-VAE Autoencoder

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	16	17	5	3	2
2	2	27	25	26	24	9	1
3	3	5	4	16	1	17	2
4	4	1	16	17	3	5	2
5	5	3	1	4	16	10	2
6	6	8	7	2	25	11	1
7	7	8	11	9	6	2	2
8	8	6	7	11	9	2	2
9	9	11	7	2	27	25	2
10	10	13	20	5	1	26	1
11	11	9	7	8	2	25	2
12	12	22	13	10	1	20	1
13	13	10	20	21	1	15	1
14	10	14	13	26	12	20	2
15	15	13	21	1	4	20	2
16	16	4	17	1	3	5	2
17	17	16	4	1	3	19	2
18	18	19	21	3	13	20	3
19	19	18	4	3	17	21	2
20	20	13	21	10	22	1	2
21	21	20	13	19	1	15	3
22	22	12	20	13	10	21	2
23	23	22	15	21	12	13	3
24	24	26	25	2	27	10	1
25	25	27	2	24	26	9	1
26	26	24	2	27	10	25	2
27	27	2	25	26	24	9	2
28	28	10	26	13	14	20	2
Total							52

Table 8-8 Same topology (R2 difference) and irrespective of source clustering results from MMD configuration 4

4) Sparse Autoencoder

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	16	21	15	2
2	2	27	25	26	24	28	1
3	3	16	5	17	28	4	2
4	4	1	17	16	15	21	2
5	5	3	10	26	20	21	2
6	6	8	7	11	9	25	1
7	7	8	11	6	9	2	2
8	8	7	6	11	9	25	2
9	9	11	7	8	27	25	2
10	10	26	20	21	12	24	1
11	11	7	8	9	6	2	2
12	12	21	20	10	26	24	2
13	13	15	20	21	26	12	1
14	14	2	24	25	12	23	1
15	15	13	21	20	12	1	1
16	16	17	4	1	3	15	1
17	17	16	4	1	3	15	1
18	18	19	28	2	27	3	1
19	19	18	28	2	27	3	1
20	20	21	12	26	10	15	2
21	21	20	12	10	15	26	1
22	22	23	24	12	25	26	1
23	23	22	12	20	21	24	1
24	24	26	25	27	10	20	1
25	25	27	24	26	2	10	1
26	26	27	24	25	10	20	2
27	27	25	26	24	2	10	1
28	28	2	1	4	3	26	1
Total							39

Table 8-9 Same topology (R2 difference) and irrespective of source clustering results from Sparse Autoencoder

Auto-encoder Type	Accuracy	DB index	Number of circuits in same cluster as Target circuit (out of first 5)
Vanilla	0.43	51.38	**53**
VAE	**0.79**	**26.75**	51
MMD	0.43	148.48	52
Sparse	0.21	385.21	39

Table 8-10 Same topology clustering performance metrics from 4 different auto-encoder architectures

8.1.3. Circuits with same source type and similar topology (R2 difference & L/C swap)

- Class 1 = {1-Vs R2 L CR, 4-Vs L CR, 3-Vs C LR, 5-Vs R2 C LR, 24-VsR2 C LR and 26-VsR2 L CR }
- Class 2 = {8-Is L CR, 11-Is R2 L CR, 7-Is C LR, 9- Is R2 C LR, 25-IsR2 C LR and 27-IsR2 L CR}
- Class 3 = {12-VsR2L R C, 13-VsR2C R L, 16-VsC L R R2, 18-VsL C R R2, 20-VsR2 C R L and 22-VsR2 L R C }
- Class 4 = {14-IsR2L R C, 15-IsR2C R L, 17-IsC L R R2, 19-IsL C R R2, 21-IsR2 C R L and 23-IsR2 L R C }
- Class 5 = {6-Vs R2 RLC and 28-Vs series RLC}
- Class 6 = {10-Is R2 RLC}

1) Vanilla Autoencoder

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	15	21	16	2
2	2	27	25	11	9	28	
3	3	16	5	4	17	1	3
4	4	1	17	16	15	21	2
5	5	3	10	1	20	21	3
6	6	8	9	7	11	25	1
7	7	8	11	9	2	27	4
8	8	7	11	9	2	27	4
9	9	11	7	8	27	25	5
10	10	26	24	20	25	27	1
11	11	8	7	9	2	27	4
12	12	21	20	15	23	1	2
13	13	20	26	24	10	25	2
14	14	2	28	25	27	11	1
15	15	21	1	4	17	12	3
16	16	17	4	1	3	15	1
17	17	4	16	1	15	21	2
18	18	19	28	2	14	21	1
19	19	18	28	4	17	21	2
20	20	10	26	24	21	13	1
21	21	15	20	12	1	23	2
22	22	23	12	20	24	14	3
23	23	21	12	15	20	1	3
24	24	26	25	27	10	20	2
25	25	27	2	24	26	9	2
26	26	24	25	27	10	20	2
27	27	25	2	9	11	26	4
28	28	2	27	25	9	11	1
Total							63

Table 8-11 Same topology(R2 difference & L/C swap) and same source clustering results from Vanilla Autoencoder

2) VAE Autoencoder

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	15	23	21	2
2	2	11	1	17	4	15	
3	3	5	19	4	17	1	3
4	4	1	17	15	23	21	2
5	5	3	19	1	4	17	4
6	1	4	17	15	23	21	0
7	5	3	19	17	1	4	0
8	2	11	17	1	4	15	1
9	5	3	17	1	4	15	0
10	10	17	15	4	1	23	1
11	11	2	14	1	15	17	1
12	12	4	1	17	15	23	1
13	13	17	4	1	15	23	1
14	14	12	24	4	17	15	2
15	15	4	17	1	23	21	3
16	16	17	4	1	15	23	1
17	17	4	1	15	23	21	3
18	18	19	13	4	1	17	2
19	19	4	1	17	15	23	3
20	20	19	1	4	17	15	1
21	21	1	4	17	15	23	3
22	22	4	17	1	15	23	1
23	23	15	17	4	1	21	3
24	24	17	4	1	15	23	3
25	24	15	23	1	17	4	0
26	26	17	4	1	15	13	3
27	26	15	23	1	17	4	0
28	3	5	19	21	4	1	0
Total							44

Table 8-12 Same topology(R2 difference & L/C swap) and same source clustering results from VAE Autoencoder

3) MMD-VAE Autoencoder

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	16	17	5	3	3
2	2	27	25	26	24	9	0
3	3	5	4	16	1	17	4
4	4	1	16	17	3	5	4
5	5	3	1	4	16	10	4
6	6	8	7	2	25	11	1
7	7	8	11	9	6	2	4
8	8	6	7	11	9	2	4
9	9	11	7	2	27	25	4
10	10	13	20	5	1	26	1
11	11	9	7	8	2	25	4
12	12	22	13	10	1	20	3
13	13	10	20	21	1	15	2
14	10	14	13	26	12	20	1
15	15	13	21	1	4	20	2
16	16	4	17	1	3	5	1
17	17	16	4	1	3	19	1
18	18	19	21	3	13	20	2
19	19	18	4	3	17	21	2
20	20	13	21	10	22	1	3
21	21	20	13	19	1	15	2
22	22	12	20	13	10	21	4
23	23	22	15	21	12	13	3
24	24	26	25	2	27	10	2
25	25	27	2	24	26	9	2
26	26	24	2	27	10	25	2
27	27	2	25	26	24	9	2
28	28	10	26	13	14	20	1
Total							68

Table 8-13 Same topology(R2 difference & L/C swap) and same source clustering results from MMD configuration 4

4) Sparse Autoencoder

Target Circuit ID	Nearby circuits in order of ascending distance to right						Number of circuits in same cluster as Target circuit (out of first 5)
1	1	4	17	16	15	21	2
2	2	27	25	28	26	24	
3	3	16	5	17	28	4	2
4	4	1	17	16	2	15	2
5	5	3	10	26	20	21	3
6	6	8	7	11	9	25	1
7	7	8	11	9	6	2	4
8	8	7	11	6	9	2	4
9	9	11	7	8	27	25	5
10	10	26	20	21	12	24	1
11	11	8	7	9	2	6	4
12	12	20	21	26	24	10	2
13	13	15	20	21	26	12	2
14	14	2	25	1	4	23	1
15	15	21	20	13	12	1	2
16	16	17	4	1	3	2	1
17	17	16	4	1	2	15	1
18	18	19	28	2	11	9	1
19	19	18	28	2	27	11	1
20	20	21	12	26	10	24	2
21	21	20	12	26	10	15	1
22	22	23	24	25	27	12	1
23	23	22	12	20	21	24	2
24	24	26	25	27	20	12	2
25	25	27	24	26	2	20	2
26	26	24	27	25	20	10	2
27	27	25	26	24	2	20	2
28	28	2	1	4	17	15	1
Total							54

Table 8-14 Same topology(R2 difference & L/C swap) and same source clustering results from Sparse Autoencoder

Auto-encoder Type	Accuracy	DB index	Number of circuits in same cluster as Target circuit (out of first 5)

Vanilla	0.43	180.92	63
VAE	**0.90**	**23.68**	44
MMD	0.61	76.25	**68**
Sparse	0.25	392.44	54

Table 8-15 Same topology(R2 difference & L/C swap) and same source clustering performance metrics from 4 different auto-encoder architectures

8.2. Implementation of 2-D Kinematic Framework in Alloy Analyser

```
//Body1d is a link

abstract sig Body{

on: one Plane,

tport: some Port,

cport: one Port,

size: one Int,

pivot: lone Port

}{tport & cport = none}

fact{all b:Body| b.size >1 &&b.size <20}// &&  integ/rem[b.size,2]=0 }

fact{all disj b, b1:Body| b.tport & b1.tport =none &&  b.cport & b1.cport =none &&  b.tport & b1.cport =none}


sig Body1d extends Body{} {#tport =2}

sig Cport in Port{}

fact{Body1d.cport in Cport }

sig Tport in Port{}

fact{ Body1d.tport in Tport}

fact{Tport & Cport = none}

abstract sig Port{

connects: lone Port,
```

connection: lone Connection

}

{Port in (Body.tport + Body.cport)}

fact{all p:Port| (p.connects&p)=none }

fact {all disj p, p1: Port | (p.connects & p1.connects) =none }// no 2 ports connect to one vice versa

fact{all b :Body| all disj p, p1:Port| (p in (b.tport + b.cport)) && p1 in (b.tport + b.cport) => p1.connects & p =none }//

fact{all disj p,p1:Port| one(p & p1.connects) => one(p1 & p.connects) }

fact{all disj b,b1: Body| b1.on & b.on.isrelativeTo = none =>((b.tport + b.cport).connects & (b1.tport + b1.cport) = none)}// makes connects symmetric

fact{all disj p,p1:Port, b,b1 :Body1d|p in (b.tport+b.cport) && p1 in (b1.tport +b1.cport) && p.connects= p1 => ((b.tport+b.cport)-p).connects & ((b1.tport+b1.cport)-p1) = none}

fact{all p: Port| p in ((rel/dom[connects] + rel/ran[connects]) & Body1d.tport) <=> p in connection.Connection } //only terminal connected ports has to have a connection

fact{all disj p,p1:Port|p.connects=p1 => one(p.connection & p1.connection) }// same connection of two ports

fact{all disj p1, p2: Port, s: State| one(tport.p1 & tport.p2) => s.updir[p1] = s.updir[p2].prevpos.prevpos && s.sec[p2]=s.sec[p1] }

sig RotTrans extends Port{}

sig Translational, Rotational extends RotTrans{}

sig Fixed extends Port{}

fact{all s:State,p: Port, b: Body| p in (b.tport + b.cport) && s.rot[p] in (Tx + Ty + Tny + Tnx) => s.rot[((b.tport + b.cport)-p)] & St = none}

fact{all p: Port, b: Body| p in (b.tport + b.cport) && p in Rotational => lone(((b.tport + b.cport)-p) & RotTrans-Rotational)}// at most one translational port can exist in bodies with rotational port

fact{all b: Body1d| lone((b.tport + b.cport) & Fixed)}//center can be stationary

//Intra body port constraints for restricting wild motion

//one port can be linearly moving other can be rotating

fact{all p:Tport, b: Body, s:State| p in b.tport && p in RotTrans-Rotational && b.cport in RotTrans-Rotational=> one(s.rot[b.cport] & s.rot[p])}//same direction of translation

fact{all s:State, p: Cport| p in RotTrans && s.rot[p] in (CCW +CW) => lone(s.rot[(cport.p).tport] & (Tx + Ty + Tny + Tnx))}

fact{all p1: RotTrans-Translational, s: State| s.rot[p1] in (CCW+CW) && p1 in Tport && (tport.p1).tport-p1 in Rotational=> s.rot[p1] = s.rot[(tport.p1).tport-p1]}// same rotation direction on both sides of body

//following constraints are derivative of length and state transitions constraints hence not used

/*

fact{all disj p1, p2: Rotational, s: State| one(tport.p1 & tport.p2) => s.rot[p2] = s.rot[p1]}

fact{all disj p1, p2: Rotational, s: State| p1 in Tport && p2 in Tport && one(tport.p1 & tport.p2) => s.rot[p2] = s.rot[p1]}// same rotation direction on both sides of body

fact{all p1: Port, s: State| s.rot[p1] in (Tx+Tnx+St) && (tport.p1).cport in Rotational && p1 in Tport => s.rot[(tport.p1).cport] = s.rot[(tport.p1).tport-p1]}//center port should also rotate in same direction as one of rotational ports

*/

abstract sig Motion{} one sig CCW, CW, Tx, Ty, Tny, Tnx, St extends Motion{}

fact{all s:State, p:Port| p in Fixed <=> s.rot[p] = St}

fact{all s:State, p:Translational| s.rot[p] & CCW = none && s.rot[p] & CW = none && s.rot[p] & St = none}

fact{all s:State, p:Rotational| s.rot[p] & Tx = none && s.rot[p] & Ty = none && s.rot[p] & Tnx = none && s.rot[p] & Tny = none && s.rot[p] & St =none}

```
fact{Translational.connects & Rotational = none && Rotational & Fixed.connects = none  }


abstract sig Connection{}

fact{all  b: Body1d| some((b.tport + b.cport).connection)}

abstract sig Driver, Constraint, Joint extends Connection{}

one sig Gear extends Constraint{}{connection.Gear in (RotTrans-Translational) }


one sig Pin, Weld, Ram extends Joint{}/

one sig Shaft, Piston extends Driver{}{connection.Shaft in (Rotational) && connection.Piston in (Translational)}

// assigns connections to port types

fact{all disj p1, p2: Port| p1.connection in (Ram+Piston) &&  p1.connects =p2 =>some( (p1+p2) &(RotTrans-Rotational)) &&  (State.rot[p1] &  (CCW+CW)) = none && (State.rot[p2] & (CCW+CW)) = none }// only closed loops allowed therefore restricting pivoting on translation ports i.e. no rotation on translating pivot

fact{all disj p1, p2: Port| p1.connection in (Pin+Shaft) &&  p1.connects =p2 => some(  (p1+p2) &(RotTrans-Translational))  }//rot trans ports with pin connection may still require use of translation motion in some state, hence pin more general than ram


//Stipulate type of relative motion between links for different connection types

fact{all disj b,b1:Body,s:State| one((b.tport+b.cport).connects & (b1.tport+b1.cport)) =>

{some((b.tport+b.cport).connection  &  Pin)  =>  {some((s.rot[b.tport+b.cport] +s.rot[b1.tport+b1.cport]) & (CW+CCW)) }else

some((b.tport+b.cport).connection  &  Ram)  => {some((s.rot[b.tport+b.cport]+s.rot[b1.tport+b1.cport]) & (Tx+Ty+Tnx+Tny)) }else

some((b.tport+b.cport).connection  &  Piston)  => {some((s.rot[b.tport+b.cport]+s.rot[b1.tport+b1.cport])& (Tx+Ty+Tnx+Tny)) } }
```

}

fact{all b:Body,s:State|some((b.tport+b.cport).connection & Shaft) => some(s.rot[b.tport+b.cport] & (CW+CCW)) }

fact{all b: Body| lone(b.tport.connection & Driver)}// only one actively driven joint allowed

// constraints to stipulate relative linear motion with ram or piston joint

//fact{all disj p1, p2: Port, s: State|p1 in (RotTrans-Rotational) && p2 in Fixed && p1.connection in (Ram+Piston) && p1.connects =p2 => one(s.rot[p2] & St) && one(s.rot[p1] & (Tx+Ty+Tnx+Tny)) }

//Following constraints constraints links to have relative translation with ram or piston connection

fact{all disj p1, p2: RotTrans-Rotational, s: State|p1.connection in (Ram+Piston) && p1.connects =p2 => (one(s.rot[p2] & s.rot[p1]) && one(s.rot[p1] & (Tx+Ty+Tnx+Tny))) // two ports can translate in same or opposite directions

or (one(s.rot[p2] & (Tnx + Tny)) && one(s.rot[p1] & (Tx+Ty))) }

fact{all disj p1, p2: RotTrans-Rotational, s: State|p1.connection in (Ram+Piston) && p1.connects =p2 && s.rot[p1] in (Tnx+Tny) && s.rot[p2] in (Tx+Ty) =>

{ one(s.rot[p2] & Tx) => one(s.rot[p1] & Tnx) } else

{one(s.rot[p2] & Ty) => one(s.rot[p1] & Tny) }

}

//gear connection constraints

fact{all disj p, p1: RotTrans-Translational, s: State| p.connects =p1 && p.connection = Gear && p in Body1d.tport && p1 in Body1d.tport => s.rot[p] = ((CCW +CW) - s.rot[p1]) }// two bodies with gear connection should rotate oppositely

fact{all disj p, p1: RotTrans-Translational, s: State| p.connects =p1 && p.connection = Gear && p in Body1d.cport && p1 in Body1d.cport => s.rot[p] = s.rot[p1] && s.cx[p]=s.cx[p1]

&& s.cy[p]=s.cy[p1]}//if two bodies have their center ports connected with Gear connection then they rotate in same direction

/two connected ports without relative translation DOF should have same x and y position even applies on gear ports

fact{all s:State, disj p, p1: Port| one(p1 & connects.p) && (s.rot[p] in (CW+CCW) or one(s.rot[p] & s.rot[p1] &St)) =>

{ p in Tport && p1 in Tport && tport.p & tport.p1 =none => {s.x[p]=s.x[p1] && s.y[p]=s.y[p1]} else

p in Cport && p1 in Tport && cport.p & tport.p1 =none => {s.cx[p]=s.x[p1] && s.cy[p]=s.y[p1]}else

p in Cport && p1 in Cport && cport.p & cport.p1 =none => {s.cx[p]=s.cx[p1] && s.cy[p]=s.cy[p1]}}

}

fact{all s:State, disj p, p1: Tport|tport.p & tport.p1 =none && p1 & connects.p = none && p.connects & p1 = none && s.x[p] =s.x[p1] => s.y[p] !=s.y[p1]}//prevent overlapping of links by avoiding two non interconnected ports of diff bodies to have same coordinates

fact{all s:State, disj p, p1: Tport|tport.p & tport.p1 =none && p1 & connects.p = none && p.connects & p1 = none && s.y[p] =s.y[p1] => s.x[p] !=s.x[p1]

/*****************************Following applies on all Bodies*************************************/

fact{all disj b,b1:Body| (b.pivot & (b1.cport + b1.tport) = none)} // no two bodies can have same pivot port

fact{all p:Fixed| p in Tport => p = (tport.p).pivot // fixed port pivots

else p = (cport.p).pivot }// by default center pivots

fact{all s:State,b:Body| some(s.rot[(b.cport + b.tport)] & (CCW+CW)) => one(b.pivot & (b.cport + b.tport)) else

#(s.rot[(b.cport + b.tport)] & (Tx+Ty+Tnx+Tny))>=2 => b.pivot & (b.cport + b.tport) = none // if any port rotational then pivot otherwise if more than two ports translational than not pivot

}

fact{all s:State, p:Port| p in Tport && one(s.rot[p] & (Tx+Ty+Tnx+Tny)) => (tport.p).pivot = p//if only one terminal tanslational then terminal pivot

else{ p in Cport && one(s.rot[p] & (Tx+Ty+Tnx+Tny)) => (cport.p).pivot = p} // if central translational then it will pivot

}

// kinematic state constraints on port position in same state

fact{all disj p1, p: Tport, s: State| one(tport.p1 & tport.p) => s.updir[p1] = s.updir[p].prevpos.prevpos && s.sec[p]=s.sec[p1] }//&& (p1 in (rel/dom[connects] + rel/ran[connects]))

fact{all disj p1, p2: Port, s: State| p1 in Body1d.tport && p2 in Body1d.cport && one(tport.p1 & cport.p2) && s.updir[p1] in (Rx+Ry)=> s.updir[p2] = s.updir[p1] && s.sec[p1]=s.sec[p2]}

fact{all p: Tport, s:State| s.updir[p] in (Rx + Ry) => s.xd[p] <= s.ncprop[(tport.p)] && s.yd[p] <= s.ncprop[(tport.p)] else

s.updir[p] in (Rnx + Rny) => s.xd[p] <= s.cprop[(tport.p)] && s.yd[p] <= s.cprop[(tport.p)]}

//boundary condition consistent with inter quarter rotation

fact{all s:State,disj p,p1: Tport|one(tport.p & tport.p1) && s.updir[p] in (Rx + Ry) && s.xd[p] = s.ncprop[(tport.p)] && s.y[p] = s.y[p1] => s.updir[p] = Ry && s.updir[p1] = Rny}

fact{all s:State,disj p,p1: Tport|one(tport.p & tport.p1) && s.updir[p] in (Rx + Ry) && s.yd[p] = s.ncprop[(tport.p)] && s.x[p] = s.x[p1] => s.updir[p] = Rx && s.updir[p1] = Rnx}

//constraints on numerical port states based on abstracted values

fact{all s:State, disj p,p1: Tport| one(tport.p & tport.p1) && s.x[p] = s.x[p1] => s.y[p1] != s.y[p] else

one(tport.p & tport.p1) && s.y[p] = s.y[p1] => s.x[p1] != s.x[p]}

fact{all s:State, disj p,p1,p2: Port| one(tport.p & tport.p2& cport.p1) && one(s.updir[p] & Rx) && one(s.updir[p1] & Rx) && one(s.updir[p2] & Rnx) => s.x[p] >= s.cx[p1] && s.cy[p1] > s.y[p] && s.cx[p1] >= s.x[p2] && s.y[p2] > s.cy[p1] }// consistent with quarter transition

fact{all s:State, disj p,p1, p2: Port| one(tport.p & tport.p2 & cport.p1) && one(s.updir[p] & Ry) && one(s.updir[p1] & Ry) && one(s.updir[p2] & Rny) => s.x[p] > s.cx[p1] && s.y[p] >= s.cy[p1] && s.cx[p1] > s.x[p2] && s.cy[p1] >= s.y[p2]}

//calculate body size in two directions w.r.t its center based on qualitative state of the port

fact{all p: Tport, s: State|s.updir[p] in (Ry+Rx) => s.x2[p] = integ/mul[s.xd[p],s.xd[p]] && s.y2[p] = integ/mul[s.yd[p], s.yd[p]] && integ/mul[s.ncprop[(tport.p)],s.ncprop[(tport.p)]] = integ/plus[s.x2[p] , s.y2[p]]}

fact{all p: Tport, s: State|s.updir[p] in (Rny+Rnx) => s.x2[p] = integ/mul[s.xd[p],s.xd[p]] && s.y2[p] = integ/mul[s.yd[p], s.yd[p]] && integ/mul[s.cprop[(tport.p)],s.cprop[(tport.p)]] = integ/plus[s.x2[p] , s.y2[p]]}

// calculate center port to terminal port L1 distance

fact{all s: State,disj p,p1: Port| p in Tport && p1 in Cport && one(tport.p & cport.p1) =>

{ s.x[p] > s.cx[p1] => {s.xd[p] = integ/minus[s.x[p],s.cx[p1]]}else

s.x[p] < s.cx[p1] => {s.xd[p] = integ/minus[s.cx[p1],s.x[p]]}else

s.x[p] = s.cx[p1] => {s.xd[p] = 0 }}

}

fact{all s: State,disj p,p1: Port| p in Tport && p1 in Cport && one(tport.p & cport.p1) =>

{ s.y[p] > s.cy[p1] => {s.yd[p] = integ/minus[s.y[p],s.cy[p1]]}else

s.y[p] < s.cy[p1] => {s.yd[p] = integ/minus[s.cy[p1],s.y[p]]}else

s.y[p] = s.cy[p1] => {s.yd[p] = 0 }}

}

fact{all s: State,p:Tport|integ/mul[s.xd[p],s.yd[(tport.p).tport-p]] = integ/mul[s.xd[(tport.p).tport-p],s.yd[p]] }// ratio between two terminal port's position components has to be same p1x/p1y = p2x/p2y

fact{all p:RotTrans-Translational, disj s, s1:State| s1= ord/next[s] && s.rot[p] = CCW => s1.updir[p] = s.updir[p] or s.updir[p].prevpos = s1.updir[p]}

fact{all p:RotTrans-Translational, disj s, s1:State| s1= ord/next[s] && s.rot[p] = CW => s1.updir[p] = s.updir[p] or s.updir[p].nextpos = s1.updir[p]}

/********** Motion Constraints**********/

//follwing are combination of Rotation directions and qualitative state of a rotating body

fact{all p:RotTrans-Translational, disj s, s1:State|p in Tport && s1= ord/next[s] && s.updir[p] = Ry && s1.updir[p] = Ry && s.rot[p]=CCW && s.yd[p] !=s.ncprop[(tport.p)] => s.xd[p] > s1.xd[p] && s.yd[p] < s1.yd[p]}// && s1.x[p] != s1.x[(port.p).port-p] && s1.y[p] != s1.y[(port.p).port-p]}

fact{all p:RotTrans-Translational, disj s, s1:State|p in Tport && s1= ord/next[s] && s.updir[p] = Ry && s1.updir[p] = Ry && s.rot[p]=CW && s.xd[p] !=s.ncprop[(tport.p)]=> s.xd[p] < s1.xd[p] && s.yd[p] > s1.yd[p]}// && s1.x[p] != s1.x[(port.p).port-p] && s1.y[p] != s1.y[(port.p).port-p]}

fact{all p:RotTrans-Translational, disj s, s1:State|p in Tport && s1= ord/next[s] && s.updir[p] = Rx && s1.updir[p] = Rx && s.rot[p]=CCW && s.xd[p] != s.ncprop[(tport.p)] => s.xd[p] < s1.xd[p] && s.yd[p] > s1.yd[p]}// && s1.x[p] != s1.x[(tport.p).tport-p] && s1.y[p] != s1.y[(tport.p).tport-p]}

fact{all p:RotTrans-Translational, disj s, s1:State|p in Tport && s1= ord/next[s] && s.updir[p] = Rx && s1.updir[p] = Rx && s.rot[p]=CW && s.yd[p] !=s.ncprop[(tport.p)] => s.xd[p] > s1.xd[p] && s.yd[p] < s1.yd[p]}// && s1.x[p] != s1.x[(tport.p).tport-p] && s1.y[p] != s1.y[(tport.p).tport-p]}

fact{all p:RotTrans-Translational, disj s, s1:State|p in Tport && s1= ord/next[s] && s.updir[p] = Rnx && s1.updir[p] = Rnx && s.rot[p]=CCW && s.xd[p] !=s.cprop[(tport.p)] => s.xd[p] < s1.xd[p] && s.yd[p] > s1.yd[p]}// && s1.x[p] != s1.x[(tport.p).tport-p] && s1.y[p] != s1.y[(port.p).port-p]}

fact{all p:RotTrans-Translational, disj s, s1:State| p in Tport && s1= ord/next[s] && s.updir[p] = Rnx && s1.updir[p] = Rnx && s.rot[p]=CW && s.yd[p] !=s.cprop[(tport.p)] => s.xd[p] > s1.xd[p] && s.yd[p] < s1.yd[p]}// && s1.x[p] != s1.x[(tport.p).tport-p] && s1.y[p] != s1.y[(.p).tport-p]}

fact{all p:RotTrans-Translational, disj s, s1:State| p in Tport && s1= ord/next[s] && s.updir[p] = Rny && s1.updir[p] = Rny && s.rot[p]=CCW && s.yd[p] !=s.cprop[(tport.p)] => s.xd[p] > s1.xd[p] && s.yd[p] < s1.yd[p]}// && s1.x[p] != s1.x[(port.p).port-p] && s1.y[p] != s1.y[(port.p).port-p]}

fact{all p:RotTrans-Translational, disj s, s1:State| p in Tport && s1= ord/next[s] && s.updir[p] = Rny && s1.updir[p] = Rny && s.rot[p]=CW && s.xd[p] !=s.cprop[(tport.p)] => s.xd[p] < s1.xd[p] && s.yd[p] > s1.yd[p]}// && s1.x[p] != s1.x[(port.p).port-p] && s1.y[p] != s1.y[(port.p).port-p]}

// intra sector transition

/*sectors not used

fact{all p:Tport, s:State| s.y[p] = s.y[(tport.p).tport-p] && s.updir[p] in Rny && s.sec[p] = 0 => s.xd[p]=s.cprop[tport.p]}

fact{all p:Tport, s:State| s.x[p] = s.x[(tport.p).tport-p] && s.updir[p] in Rx && s.sec[p] = 0=> s.yd[p]=s.ncprop[tport.p]}

*/

//early entrance to next quarter when going CCW but exact entrance when CW i.e. always quarter towards CCW is preferred

fact{all p:RotTrans-Translational, disj s, s1:State, b: Body| p in b.tport && s1= ord/next[s] && one(s.rot[p]&CCW) && one(s.updir[p] & Rx) && (integ/minus[s.y[(b.tport-p)],s.y[p]]<2 or s.xd[p] = s.ncprop[(tport.p)])=> one(s1.updir[p] & Ry)}

fact{all p:RotTrans-Translational, disj s, s1:State, b: Body| p in b.tport && s1= ord/next[s] && one(s.rot[p]&CCW) && one(s.updir[p] & Ry) && (integ/minus[s.x[p],s.x[(b.tport-p)]]<2 or s.yd[p] =s.ncprop[(tport.p)]) => one(s1.updir[p] & Rnx) }

fact{all p:RotTrans-Translational, disj s, s1:State, b: Body| p in b.tport && s1= ord/next[s] && one(s.rot[p]&CCW) && one(s.updir[p] & Rnx) && (integ/minus[s.y[p],s.y[(b.tport-p)]]<2 or s.xd[p] =s.cprop[(tport.p)]) => one(s1.updir[p] & Rny)}

fact{all p:RotTrans-Translational, disj s, s1:State, b: Body| p in b.tport && s1= ord/next[s] && one(s.rot[p]&CCW) && one(s.updir[p] & Rny) && (integ/minus[s.x[(b.tport-p)],s.x[p]]<2 or s.yd[p] =s.cprop[(tport.p)])=> one(s1.updir[p] & Rx)}

fact{all p:RotTrans-Translational, disj s, s1:State, b: Body| p in b.tport && s1= ord/next[s] && one(s.rot[p]&CW) && one(s.updir[p] & Ry) && s.y[p]=s.y[(b.tport-p)] => one(s1.updir[p] & Rx)}

fact{all p:RotTrans-Translational, disj s, s1:State, b: Body| p in b.tport && s1= ord/next[s] && one(s.rot[p]&CW) && one(s.updir[p] & Rx) && s.x[p]=s.x[(b.tport-p)] => one(s1.updir[p] & Rny)}

fact{all p:RotTrans-Translational, disj s, s1:State, b: Body| p in b.tport && s1= ord/next[s] && one(s.rot[p]&CW) && one(s.updir[p] & Rny) && s.y[p]=s.y[(b.tport-p)] => one(s1.updir[p] & Rnx

fact{all p:RotTrans-Translational, disj s, s1:State, b: Body| p in b.tport && s1= ord/next[s] && one(s.rot[p]&CW) && one(s.updir[p] & Rnx) && s.x[p]=s.x[(b.tport-p)] => one(s1.updir[p] & Ry)}

// non connected ports of a body with connected ports cannot have same angular velocity

fact{all disj p,p1:Port,s:State| p.connects = p1 && p.connection = Pin => // any type can have this connection thats y connection domain type ommited above

{p in Tport && p1 in Tport =>{ s.w[((tport.p).tport-p)] & s.w[((tport.p1).tport-p1)] = none }else// center velocity if attached together can be same

p in Cport && p1 in Cport =>{ s.w[((cport.p).tport)] & s.w[((cport.p1).tport)] = none }else// center velocity if attached together can be same

p in Tport && p1 in Cport =>{ s.w[((tport.p).tport-p)] & s.w[((cport.p1).tport)] = none }}// center velocity if attached together can be same

}

/if only two ports are translational

fact{all p:RotTrans-Rotational, disj s,s1:State| s.rot[p] in (Tx + Tnx + Tny + Ty) && s.rot[(tport.p).cport] in (Tx + Tnx + Tny + Ty) && s1=ord/next[s] => s.xd[p] = s1.xd[p] && s.yd[p] = s1.yd[p] && s.updir[p] = s1.updir[p] && s.sec[p] = s1.sec[p] }

fact{all p: RotTrans-Rotational, disj s, s1:State|p in Cport && s1 =ord/next[s] && one(s.rot[p] & Tx) => s1.cx[p] >s.cx[p] && s1.cy[p] = s.cy[p]}

fact{all p: RotTrans-Rotational, disj s, s1:State|p in Cport && s1 =ord/next[s] && one(s.rot[p] & Ty) => s1.cx[p] = s.cx[p] && s1.cy[p]>s.cy[p] }

fact{all p: RotTrans-Rotational, disj s, s1:State|p in Cport && s1 =ord/next[s] && one(s.rot[p] & Tnx) => s1.cx[p]<s.cx[p] && s1.cy[p] = s.cy[p] }

fact{all p: RotTrans-Rotational, disj s, s1:State|p in Cport && s1 =ord/next[s] && one(s.rot[p] & Tny) => s1.cx[p] = s.cx[p] && s1.cy[p] <s.cy[p] }

fact{all p: RotTrans-Rotational, disj s, s1:State|p in Tport && s1 =ord/next[s] && one(s.rot[p] & Tx) => s1.x[p] >s.x[p] && s1.y[p] = s.y[p]}

fact{all p: RotTrans-Rotational, disj s, s1:State|p in Tport && s1 =ord/next[s] && one(s.rot[p] & Ty) => s1.x[p] = s.x[p] && s1.y[p]>s.y[p] }

fact{all p: RotTrans-Rotational, disj s, s1:State|p in Tport && s1 =ord/next[s] && one(s.rot[p] & Tnx) => s1.x[p]<s.x[p] && s1.y[p] = s.y[p] }

fact{all p: RotTrans-Rotational, disj s, s1:State|p in Tport && s1 =ord/next[s] && one(s.rot[p] & Tny) => s1.x[p] = s.x[p] && s1.y[p] <s.y[p] }

//relative velocity if travelling in same direction

fact{all disj p,p1:Port, disj s, s1:State| s1= ord/next[s] && p.connects = p1 && p.connection = Ram =>

{s.rot[p] in Tx && s.rot[p1] in Tx =>{ s.vx[p] & s.vx[p1] = none }else

s.rot[p] in Ty && s.rot[p1] in Ty =>{ s.vy[p] & s.vy[p1] = none }}

}

fact{all p: Port, disj s,s1: State| s1=ord/next[s] && s.rot[p]=St =>

{p in Tport => {s.x[p]=s1.x[p] && s.y[p]=s1.y[p]}else

p in Cport => {s.cx[p]=s1.cx[p] && s.cy[p]=s1.cy[p]}}

}

//if two ports are welded then other non welded ports of respective bodies should have have same motion direction and same angular velocity i.e. no relative motion between two links

fact{all disj p,p1:Port,s:State| p.connects = p1 && p.connection = Weld =>

{p in Tport && p1 in Tport =>{ s.w[((tport.p).tport-p)] = s.w[((tport.p1).tport-p1)] && one(s.rot[((tport.p).tport-p)] &s.rot[((tport.p1).tport-p1)]) }else

p in Cport && p1 in Cport =>{ s.w[((cport.p).tport)] = s.w[((cport.p1).tport)] && one(s.rot[((cport.p).tport)] &s.rot[((cport.p1).tport)]) }else

p in Tport && p1 in Cport =>{ s.w[((tport.p).tport-p)] = s.w[((cport.p1).tport)] && one(s.rot[((tport.p).tport-p)] &s.rot[((cport.p1).tport)]) }

sig State{

updir: Port -> Dir,

sec: Port -> one Int,

rot: Port -> Motion,

xd: Tport -> one Int,

yd: Tport -> one Int,

x2: Tport -> one Int,

y2:Tport -> one Int,

cx: Cport -> one Int,

cy: Cport -> one Int,

x: Tport -> one Int,

y: Tport -> one Int,

vx: Port -> one Int,

vy: Port -> one Int,

v: Port -> one Int,

w: Port -> one Int,

```
cprop: Body1d -> one Int,

ncprop: Body1d -> one Int

}

fact{all s: State, p: Port| p in Body1d.tport =>

p in rel/dom[s.xd] && p in rel/dom[s.yd] &&

p in rel/dom[s.x2] && p in rel/dom[s.y2] &&

 p in rel/dom[s.x] && p in rel/dom[s.y] &&

 one(p.(s.x)) && one(p.(s.y)) &&

one(p.(s.x2))&& one(p.(s.y2)) &&

one(p.(s.sec))  && one(p.(s.xd))&& one(p.(s.yd))&&

s.x2[p]<=121 && s.y2[p]<=121  && s.xd[p]<=11 && s.yd[p]<=11 &&  s.xd[p]>=0 && s.yd[p]>=0 && s.x2[p]>=0 && s.y2[p]>=0 &&

s.x[p]>=0 && s.y[p]>=0

}

fact{all s:State, p:Port| p in rel/dom[s.w] && p in rel/dom[s.vx] && p in rel/dom[s.vy] && p in rel/dom[s.v] && p in rel/dom[s.updir] &&  p in rel/dom[s.sec] && p in rel/dom[s.rot] &&

one(p.(s.w))  && one(p.(s.v))  && one(p.(s.vx))  && one(p.(s.vy))  && one(p.(s.updir))  && one((p).(s.rot))  &&  one((p).(s.sec))  && s.w[p]>=0 && s.w[p]<=10 && s.v[p]>=0 && s.v[p]<=11 && s.vx[p]<=10 && s.vy[p]<=10 && s.vx[p]>=0&& s.vy[p]>=0}

fact{rel/dom[State.x] & Body1d.cport = none && rel/dom[State.y] & Body1d.cport = none

&& rel/dom[State.cx] & Body1d.tport = none && rel/dom[State.cy] & Body1d.tport = none}

fact{all s: State, p: Cport|p in rel/dom[s.cx] && p in rel/dom[s.cy]  && one(p.(s.cx)) && one(p.(s.cy)) && s.cx[p]>=0 && s.cy[p]>=0}



fact{all s:State, b:Body1d | s.sec[(b.tport + b.cport)]>=0 && s.sec[(b.tport + b.cport)]<3 &&
```

b in rel/dom[s.cprop] && one(b.(s.cprop)) && s.cprop[b] <= 11 && s.cprop[b] >0 && b in rel/dom[s.cprop] && one((b).(s.cprop)) &&

b in rel/dom[s.ncprop] && one(b.(s.ncprop)) && s.ncprop[b] <=11 && s.ncprop[b] >0 && b in rel/dom[s.ncprop] && one((b).(s.ncprop)) &&

s.ncprop[b] = integ/minus[b.size,s.cprop[b]] // body length calculation w.r.t its center towards each direction

}

//calculate velocity components

fact{all disj s,s1:State, p:Tport| s1 =ord/next[s] && s.x[p] > s1.x[p] => s.vx[p]=integ/minus[s.x[p],s1.x[p]] else

s1.x[p] > s.x[p] => s.vx[p]=integ/minus[s1.x[p],s.x[p]] else

s.x[p] = s1.x[p] => s.vx[p]= 0

}

fact{all disj s,s1:State, p:Tport| s1 =ord/next[s] && s.y[p] > s1.y[p] => s.vy[p]=integ/minus[s.y[p],s1.y[p]] else

s1.y[p] > s.y[p] => s.vy[p]=integ/minus[s1.y[p],s.y[p]] else

s.y[p] = s1.y[p] => s.vy[p]= 0

}

fact{all disj s,s1:State, p:Cport| s1 =ord/next[s] && s.cx[p] > s1.cx[p] => s.vx[p]=integ/minus[s.cx[p],s1.cx[p]] else

s1.cx[p] > s.cx[p] => s.vx[p]=integ/minus[s1.cx[p],s.cx[p]] else

s.cx[p] = s1.cx[p] => s.vx[p]= 0

}

fact{all disj s,s1:State, p:Cport| s1 =ord/next[s] && s.cy[p] > s1.cy[p] => s.vy[p]=integ/minus[s.cy[p],s1.cy[p]] else

s1.cy[p] > s.cy[p] => s.vy[p]=integ/minus[s1.cy[p],s.cy[p]] else

s.cy[p] = s1.cy[p] => s.vy[p]= 0

```
}

fact{all disj s,s1:State, p:Port|  s1 =ord/next[s] && integ/plus[s.vx[p],s.vy[p]] = 1 => s.v[p] =
1 else

s.v[p] = integ/div[integ/plus[s.vx[p], s.vy[p]],2] // total velocity = average of two components

}

//Angular velocity calculation with different radiaii with factor of 10 as only integers are used

fact{all s:State, p:Tport|  s.updir[p] in (Rx+Ry) && (tport.p).tport-p = (tport.p).pivot => s.w[p]
=  integ/div[integ/mul[s.v[p],10],(tport.p).size]  &&  s.w[(tport.p).cport]  =
integ/div[integ/mul[s.v[(tport.p).cport],10],s.cprop[(tport.p)]] else

 s.updir[p]  in  (Rx+Ry)  &&  (tport.p).cport  =  (tport.p).pivot  =>  s.w[p]  =
integ/div[integ/mul[s.v[p],10],s.ncprop[(tport.p)]] else

s.updir[p]  in  (Rnx+Rny)  &&  (tport.p).tport-p  =  (tport.p).pivot  =>  s.w[p]  =
integ/div[integ/mul[s.v[p],10],(tport.p).size]  &&  s.w[(tport.p).cport]  =
integ/div[integ/mul[s.v[(tport.p).cport],10],s.ncprop[(tport.p)]]  else

 s.updir[p]  in  (Rnx+Rny)  &&  (tport.p).cport  =  (tport.p).pivot  =>  s.w[p]  =
integ/div[integ/mul[s.v[p],10],s.cprop[(tport.p)]]

}

abstract sig Dir{

nextpos: one Dir,

prevpos: one Dir,

amount: one Int

}

fact{all d: Dir| prevpos = ~nextpos &&  (d.nextpos & d) =none}// && (d.nextpos & d1) = none}

 one sig Rnx, Rx, Ry, Rny extends Dir{}


sig Plane{

haspos: seq Dir,
```

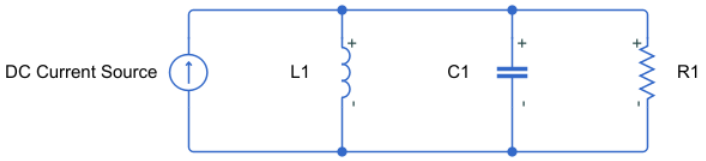
```
isrelativeTo: set Plane,

over: one Body

}

fact{over=~on}

fact{all pl:Plane| pl.isrelativeTo & pl = none}

fact{all disj pl,pl1: Plane| one(pl & pl1.isrelativeTo) => one( pl1 & pl.isrelativeTo)

fact{all disj b, b1: Body| b.on & b1.on =none}

fact{all nx: Rnx,x: Rx,y: Ry,ny: Rny, p:Plane |p.haspos =(0 ->nx) +  (1->y) + (2->x) +(3->ny)
}

fact{all disj i,j: Int| all p: Plane| (j = integ/plus[i,1]  && j>=1 && i>=0 && j<=3 && i<=2) =>
(i.(p.haspos)).nextpos =j.(p.haspos) && (3.(p.haspos)). nextpos = 0.(p.haspos)}

fun sqrt[i: Int, j:Int]: one Int{

 integ/plus[(integ/mul[i, i]), (integ/mul[j, j]) ]

}

pred possub[x: Int,y: Int, z: Int ]{

x>y => integ/minus[x,y]=z

else y>x => integ/minus[y,x]=z

else x=y => 0=z

}

pred possub2[x: Int,y: Int, i: Int,j: Int]{

x>y =>   x = i && y = j

else y>x => y = i && x = j

else x=y => x = i && x= j

}
```

pred ex{ Port.connection & Gear = none && Fixed & rel/dom[connects] = none && #Tport.connects>2 && Tport.connects & Cport =none && one(plord/first.over.tport & Fixed) && one(plord/last.over.tport & Fixed) && ord/first.y[(plord/first.over.tport & Fixed)] = ord/first.y[(plord/last.over.tport & Fixed)] }

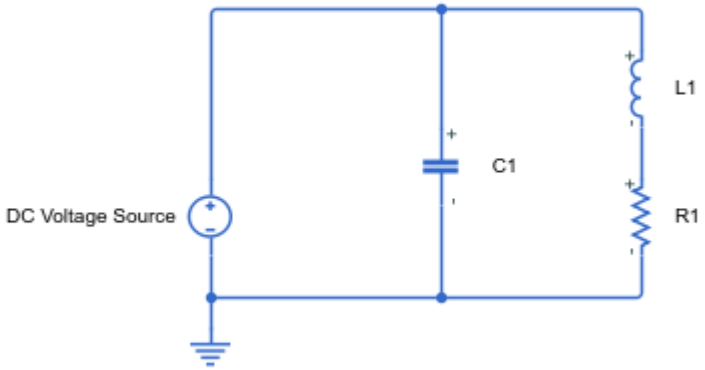
run ex for 8 but 0..127 Int, 3 Body1d, 9 Port,2 State, 3 Plane, 5 seq, 4 Dir

8.3. List of Manually Designed Circuits

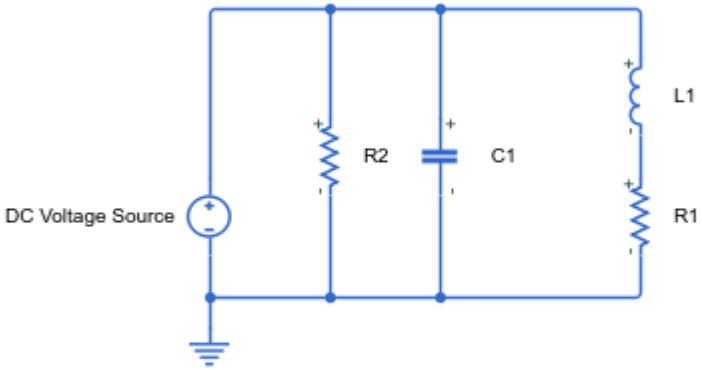
2) Is parallel RLC



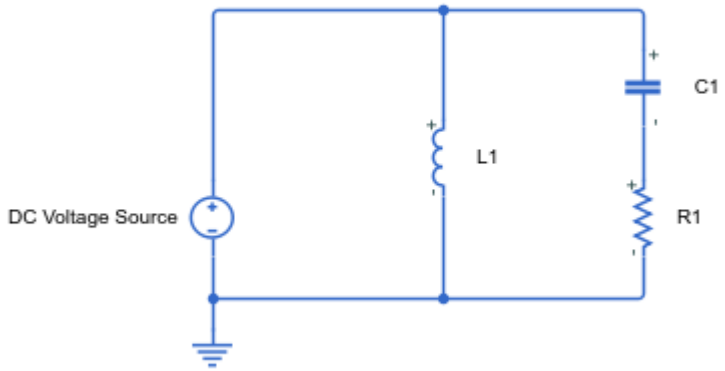
3) Vs C LR and 7) Is C LR



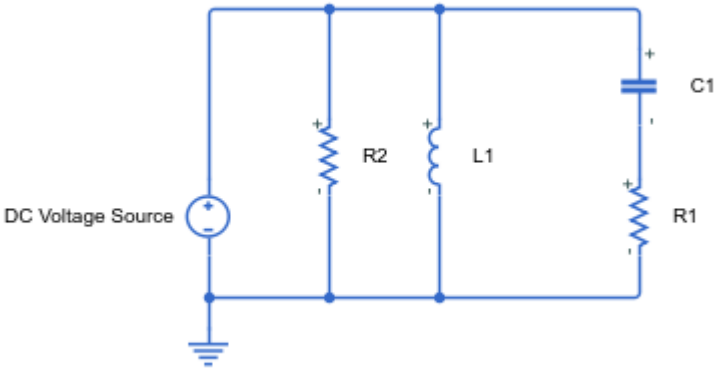
5) Vs R2 C LR and 9) Is R2 C LR



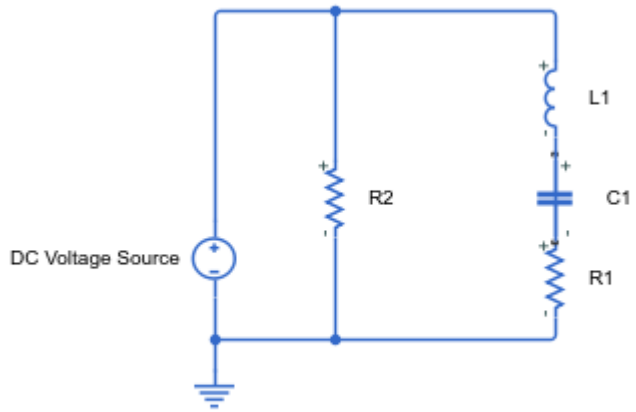
4) Vs L CR and 8) Is L CR



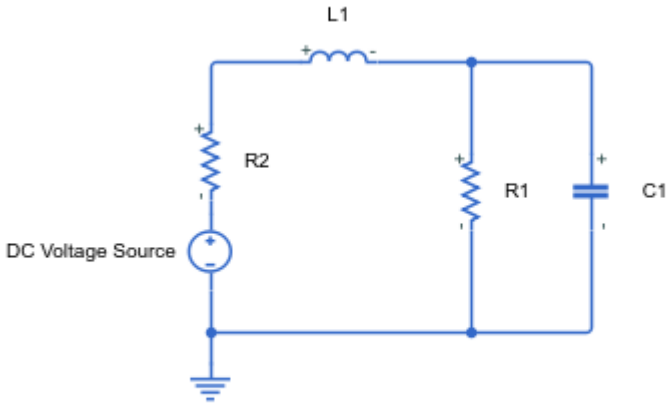
1) Vs R2 L CR and 11) Is R2 L CR



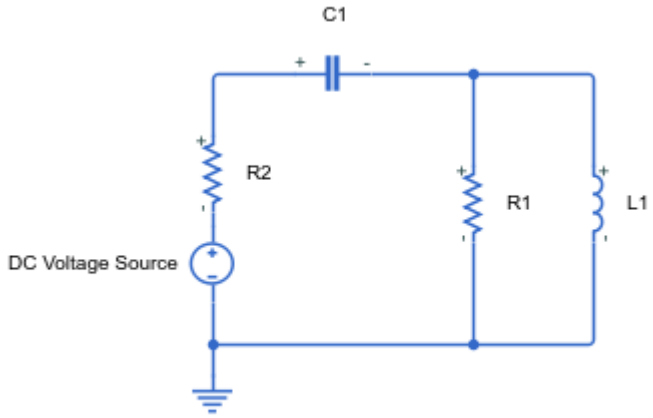
6) Vs R2 RLC and 10) Is R2 RLC



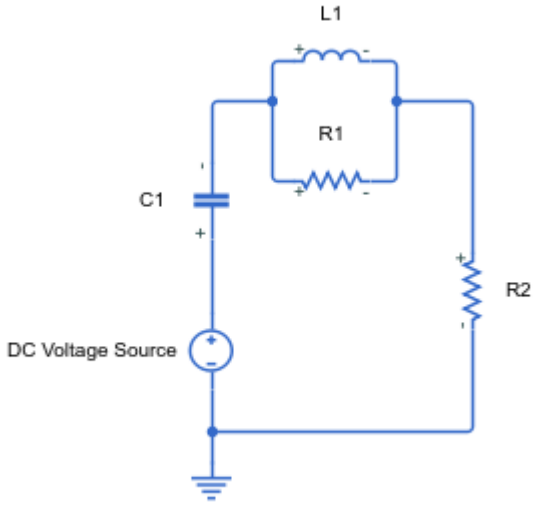
12) VsR2L R C and 14) IsR2L R C



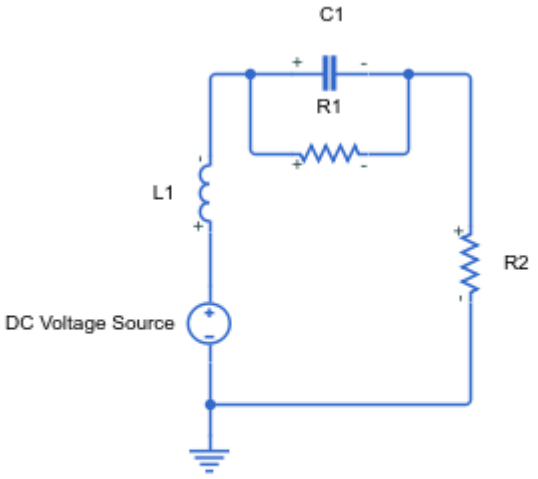
13) VsR2C R L and 15) IsR2C R L



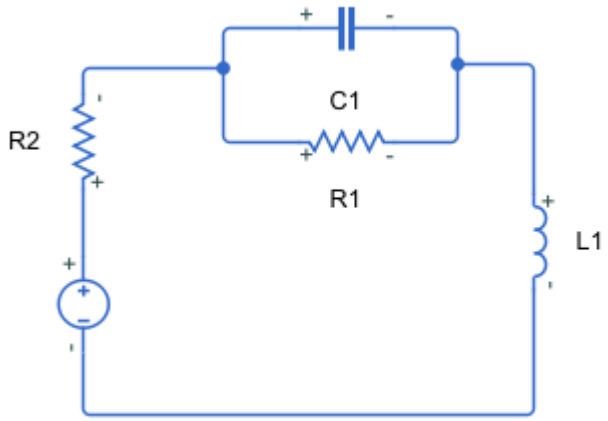
16) VsC L R R2 and 17) IsC L R R2



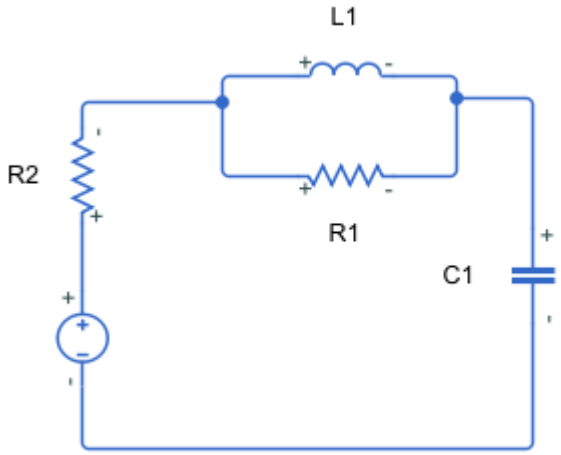
18) VsL C R R2 and 19) IsL C R R2



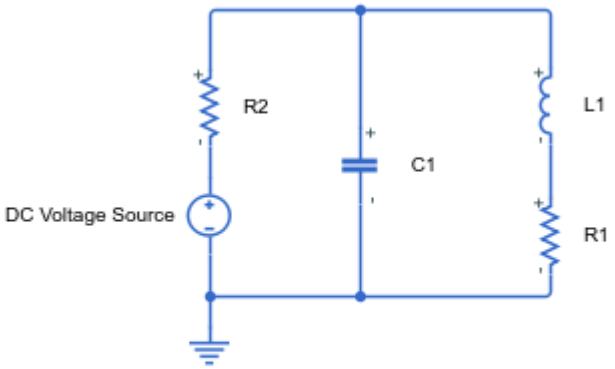
20) VsR2 C R L and 21) IsR2 C R L



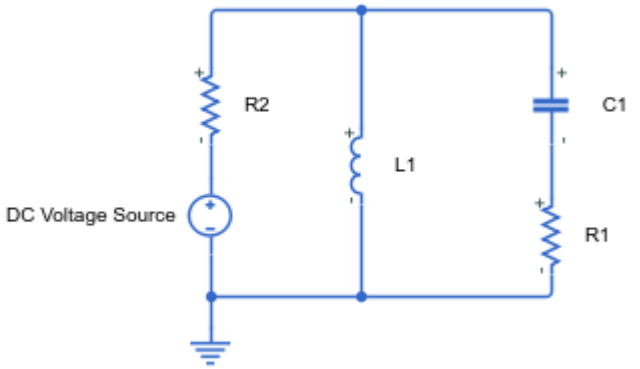
22) VsR2 L R C and 23) IsR2 L R C



24) VsR2 C LR and 25) IsR2 C LR



26) VsR2 L CR and 27) IsR2 L CR



8.4. MBSE Methodologies Overview

Sections from 8.4.1. to 8.4.3. are adopted from survey on MBSE methodologies [33] whereas section 8.4.4. is adopted from a brief introduction given on ARCADIA methodology and its supporting Capella[38] tool [6].

8.4.1. Harmony SE Methodology

Telelogic's DOOR and Harmony SE methodology is tool neutral meaning it can be adopted by any tool applying MBSE. This methodology requires use of a centralised depository for system model maintenance. Harmony is implemented using SysML and follows service request driven modelling approach. In this approach communication between modelling activities is

[38] https://polarsys.org/capella/

conducted by depicting each modelling activity's provided and required services (modelling artefacts such as test cases, interface control document etc.).

This methodology entails requirement analysis, system functional analysis and architectural design phases (shown in Figure 8-2) which are conducted during system design whereas its integration, verification and validation phases are not discussed here.

Requirement analysis phase involves:

- Elicitation and derivation
- Identification of system states and modes
- System functionality/modes allocation to physical architecture i.e. components

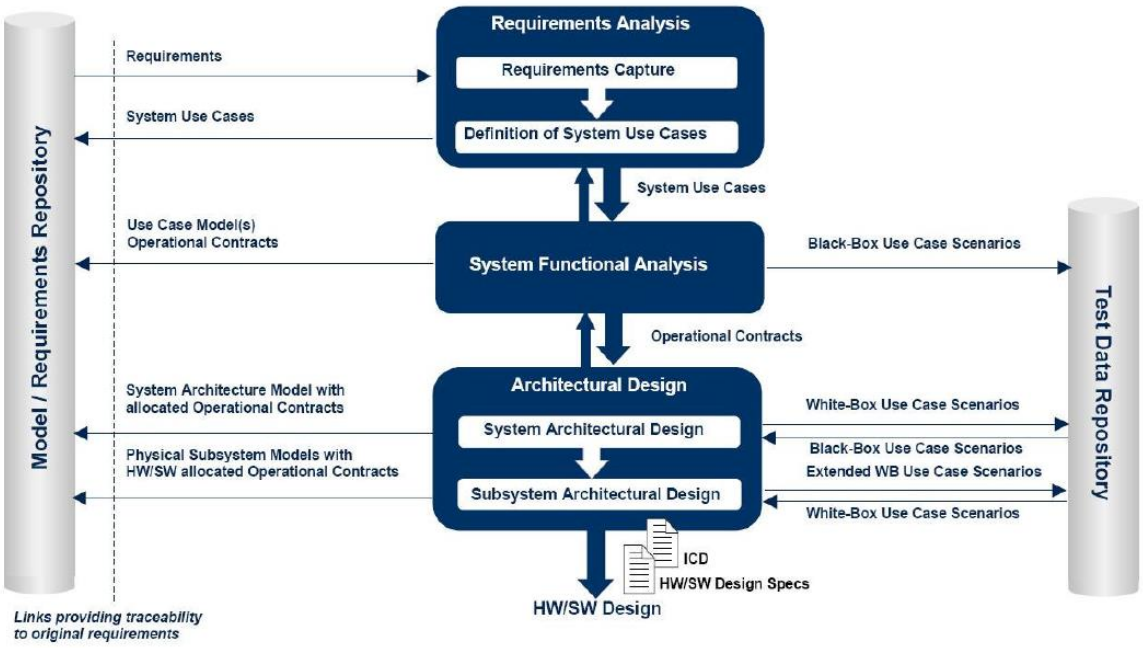


Figure 8-2 Workflow of activities in Harmony-SE methodology [33]

System functional analysis is performed by defining use cases derived in requirement analysis using BDD and IBD. The use case is built using high level sequence, activity and state chart diagrams before their verification and validation. All the use cases are then combined into a use case collaboration model which is then verified for integration consistency i.e. whether each use case provides correct inputs and outputs to other. Finally merger of use cases takes place in a system block represented using unique operational contract definitions. These operational contracts (artefact of functional analysis i.e. use cases) represents system functionality in terms of system's state/mode changes during various operations.

Architectural design starts from subsystem architecture design which is modelled using BDD and IBD. Then using operational contracts derived in functional analysis, functionality mandated in operational contracts is allocated to subsystem/components using activity diagrams, this step is also called operational contract allocation. Elaboration of each of the function takes place using Sequence diagrams along with definition of ports and interfaces based on component interactions depicted in sequence diagrams. After repeating these steps for each use case. State based behaviour is defined for each subsystem using state chart diagrams. Finally the architectural model is verified and validated.

This methodology is supported by tools like Tau and Rhapsody from Telelogic AB, however no framework to integrate analysis and design tools with modelling tool is available.

8.4.2. Object Oriented System Engineering Methodology

This methodology combines traditional system engineering methods for top down system development with model based approach to specify, analyse, design and verify systems while retaining flexibility and extensibility to tackle requirement changes. It also intends to close integration gap between object oriented software and hardware development. This methodology is very broad as it also takes into account organisational structure and mission analysis beside the technical activities of system engineering process.

This methodology is characterised by its features such as integrated product development and recursive Vee lifecycle process model applicable at multiple levels of system hierarchy. It includes following development activities

- Analyse Stakeholder Needs
- Define System Requirements
- Define Logical Architecture
- Synthesize Candidate Allocated Architectures
- Optimize and Evaluate Alternatives
- Validate and Verify System

Successful achievement of each activity required execution of supporting tasks like configuration management, risk management, monitoring and quality assessment.

It advocates model/artefact reuse for product line development. As SysML is primary language used in artefact generation, the consistency between various model views can also be ensured.

Stakeholder needs analysis: This activity captures current and perceived enterprise and system (if it is being reengineered) state. Limitation of current state are derived using causal analysis and required mission and enterprise model is developed from them. An enterprise model is used

to represent the organisation, the system to be developed and related stakeholders. The mission requirements are given in terms of mission objectives, effectiveness measures and higher level use cases which represent enterprise functionality.

System Requirements Definition: System level requirements and use cases scenarios are derived from mission level objectives to represent system level interaction with users and system operations respectively. The scenarios are represented separately from use cases using activity diagram and are used to derive functional, interface, informational and performance requirements. Probability associated with requirement changes along with potential risks resulting from such changes are used for requirement variation analysis.

Logical architecture definition: In this activity system is decomposed into functional partitions called logical components. This architecture is implementation or platform independent and only entails system functionality that can be achieved by multiple implementation solutions e.g. user interface can be of various types using knobs or buttons or touch screens etc. Logical architecture is defined using logical scenarios defined in previous activity. The system functionality can be divided in multiple ways depending on criteria such as reliability, design for change etc., to form logical components.

Candidate allocated architectures synthesis: The functionality distribution or mapping of logical components onto hardware and software components or system nodes (which defines resource distribution) is undertaken depending on partitioning criteria which take into account distribution concerns such as performance, security etc. This relationships between system components established by this distribution defines various architectures such as data, H/W and S/W.

Alternative Evaluation and Optimisation: Parametric models of candidate architectures are built to perform trade studies and analyses (in order to optimise the selected alternative) in terms of concerns, such as cost, performance etc. Potential risks are also identified by estimating resources required to achieve required performance measures set out in requirement analysis activity.

System verification and validation: System functional verification and system validation is executed in this activity by developing verification plans, test cases etc., using use cases, scenarios and design specifications. The verification plans and test cases can also be modelled using activity or any other relevant SysML diagrams suitable for system operation representation. The results of test cases are traced back to requirements stored in requirements database for checking conformance.

No dedicated tool support available but general system modelling tools which uses SysML can be used along with requirement management tools like DOORS.

8.4.3. Vitech MBSE Methodology

This methodology is based on four core concurrent SE activities (Source requirement analysis, Behaviour analysis, Architecture Synthesis and Design V&V) linked to a common System Design Repository and they are individually associated with separate domains namely, Requirements, Behaviour, Architecture and V&V domain.

This methodology follows 4 principles which are;

- Usage of modelling language and semantically formalised diagrams for problem and solution definition.
- Usage of central design repository
- Perform horizontal system development before going vertically i.e. work on all system development activity concurrently of the development process at higher level before going into detail.
- Apt tool usage especially for tasks like documentation and report generation.

The emphasis on horizontal integration over vertical is supported by the adoption of 'Onion' model in which deeper layers start becoming more detailed but backward iterations are also allowed. It is claimed that this model is advantageous over the waterfall model in that it provides overall system model (relations between all activities) at start of development process, hence provide benefit of early risk identification. This model also has some work flow restrictions due to concern of design convergence, one of which is that backward iteration of only one layer is allowed otherwise the requirements or the higher level design must be changed to allow constraint relaxation. A criteria to check completion of every activity upto required detail at each layer is also provided.

Each layer is allocated timelines to schedule the development activities for either moving top down to more detail or bottom up for modifications (reverse engineering).

Guidance to achieve top level activities is provided by associating learning objectives with activities and sub-activities e.g. the objective for performing source requirement and analysis activity is to identify requirements structure and to analyse them whereas to complete this activity it is necessary to execute following sub-activities;

- Requirement elicitation
- Requirement organisation
- Requirement analyses for identifying issues, such as conflicts and contradictions, and risks

- Creation of requirements relationships

The function/behaviour analysis activity is supported by enhanced function flow block diagrams as well as with activity diagram of SysML (provided in latest version of supporting tool Vitech CORE) to represent control flow, data interaction and function sequencing. The Design Verification and Validation is supported by test plans and test paths (pertaining to test threads) developed during requirement elicitation sub-activity and during system behaviour analysis activity respectively. Tests of types such as functional (black-box type), structural (under criteria of security, safety, operational availability etc.), performance (under nominal operating range), recovery (under different failure modes), interface (object and control flow under various conditions) and stress (under non-nominal operating range) can be executed using methods and tools associated with methodology.

Vitech advocates for single MBSE System Definition Language (SDL) to unambiguously model artefacts based on conceptual data model, built using standardised schema or ontology for explicitly defining semantics and syntax of the model artefacts. It is claimed that it will help to guide developers in requirement analysis and system design by providing common terminology and it also provide a standard input to viewpoint and documentation generation while enabling automatic consistency checking.

Developers of this methodology emphasise on development of three core models namely, control (function/behaviour), interface and physical, to derive design specifications that can be achieved by available technical means.

8.4.4. ARCADIA (Architectural Analysis and Design Integrated Approach)

This methodology has been developed by Thales by utilising experience of modelling experts of various domains in developing complex systems and is industry tested due to its implementation in Capella (graphical modelling tool). It is mostly based on functional analysis and function allocation to actual components.

An associated domain specific language has been built using or repurposing concepts (diagrams along with their elements) from UML/SysML and NAF (NATO Architecture Framework) standards to serve variety of system development needs in domains such as space, transportation etc. It places emphasis on separation between concerns/needs (operation and system needs) and solution (expressed using logical and physical architectures) while following IEEE 1220 standard. It mandates three core activities to achieve full-fledged MBSE which are modelling and analysis of need, requirement engineering and building and validation of architecture.

This methodology has extended traditional requirement engineering by inclusion of operational need analysis where user's expected way of usage and usage conditions are described along with required integration, verification, validation and qualification conditions. Whereas in system need analysis required behaviour of the system and its external interfaces are determined.

The operational need analysis is performed using operational architecture blank diagram, which is used to relate entities involved in operational scenario, such as high level functions and users (participating systems), to operational functions or activities using object flow and message flow.

The system need analysis is performed by using dataflow diagrams, which depicts relations between functions, information flow and also allows to categorize information flow using colour coding, and by using system architecture blank diagram which is used for function to component allocation using similar elements as last two types of diagrams but in addition provide elements to represent interfaces apart from data and object flow.

The next activity is to organise the system and derive its associated logical architecture in parallel by balancing operational requirements with non-functional constraints. The building of logical architecture involves organising logical (functional) components and modelling interactions between them based on the trade-offs between non-functional constraints (i.e. operational usage bounds) and feasibility of actual system realisation based on available technology.

Sequence diagram can be used in logical architecture development to model interface scenarios, functional scenarios and exchange scenario by using lifelines which represents actors or functions and message interactions which represents exchange items (functions, object, information). Different scenarios can be used to represent different viewpoints associated to concerns such as real time functionality, safety, security, functionality consistency etc. Same applies to use of any other diagram in this phase.

Finally, the physical architecture is derived to enable IVVQ (Integration Verification Validation Qualification) to identify development and technical issues associated to each separate concerns being addressed by the system to be developed. The physical architecture represents system components (hardware and software) using tree diagrams (called physical functional breakdown diagram) in hierarchical breakdown manner and can be used to evaluate development and implementation success.

All of described modelling activities takes into account the issues related to implementation and relate it to the conceptualised system function/behaviour and system structure as well as to the customer needs.

All the diagrams related to different activities can be used in any phase of the development to suite the task at hand such as using functional chains and scenarios to support capability based model organisation, for functional allocation to resources etc.

8.5. Design Patterns Representation and Retrieval Techniques

Different formats can be used to represent designs patterns [51] e.g. abstract syntax tree for structure aspect (for structure modelled in BDD/IBD), matrix (N2 charts), graph grammars, ontologies (abstract semantic graph) for semantic aspect, XML, Control flow graphs for behaviour aspects (activity diagram), (UML class diagram conversion to) directed graph (through conversion chain of Eulerian circuit to string to bit vector), temporal logic of Action to represent dynamic aspect etc. Matrices can be used in predicate matching by representing different relation between components like in interface matrices in system engineering.

Case identification algorithms e.g. Bit vector, grammar rules etc., can be applied on these representations to match problem pattern with stored patterns. Two types of matching techniques, namely Exact and Approximate matching can be used for solution retrieval. Exact matching is performed with aim to identify an architectural model with behaviour and structure matching the query pattern, normally describe in form of rules. Approximate matching algorithms calculate similarity scores based on certain structural features of the stored case and are used on representation such as matrices. Any retrieved case with similarity score above a set threshold value is selected. The template matching, for approximate matching, is based on correlating graph representation to estimate degree of similarity. Bayesian matching can also be used with query as condition. Other machine learning algorithms for pattern mining can also be used. Selection of suitable algorithm is based on precision of matching and recalling (scope of discovery i.e. how many right matches are discovered, which is based on effectiveness of learning).

8.6. Python Code Implementing MMD-VAE with Tensorflow

```
import tensorflow as tf

import numpy as np

from matplotlib import pyplot as plt
```

```
import math, os

#from tensorflow.examples.tutorials.mnist import input_data

#os.environ["CUDA_VISIBLE_DEVICES"]="-1"  # if GPU not to be used


import scipy.io as spio

import random

import csv

from numpy import inf

from sklearn import manifold

from sklearn.metrics import mean_squared_error

from utils import *


from keras.layers import regularizers

from keras.layers.advanced_activations import LeakyReLU

training_epochs =1001

exc_prop = 3# these are the number of component property values to exclude from input data RLC values


lr_rate = 1e-4

BATCH_SIZE = 72

total_circuits = 100 # number of random circuit topologies

num_signals = 5

signal_len = 1000

sample_size = 12;
```

```
total_sets = 28; # number of test circuits

trial_size = signal_len*num_signals

n_samples = 6

recall_save = False # recall previously save model

save_epochs=500#has to be greater than batch size

ISOMAP_dim=250 #isomap embedding dimension

z_dim1 = 3

z_dim2 = 1000

dec_hidden = 128

enc_hidden=128

norm_scaler = 1.0 # normalisation scaling and shifting

norm_shifter = 0.

norm_axis = -1

weight_mmd = 1.0

alpha=0.2 # for lrelu

lrelu = LeakyReLU(alpha)

enc_act = "linear"

dec_act = "linear"

term_act="sigmoid"

mid_act=tf.identity

beta = 0.001 # for l2 regulariser


def encode (train_queue, reuse=False):

    with tf.variable_scope('encode') as vs:
```

```
        if reuse:
            vs.reuse_variables()
 lrelu = LeakyReLU(alpha = alpha)
causal_conv = tf.keras.layers.Conv1D(enc_hidden, 2,1,padding="causal",activation = enc_act, kernel_regularizer=regularizers.l2(beta), name= "encode_Y_c")
        causal_conv1 = tf.keras.layers.Conv1D(enc_hidden//2, 2,1,padding="causal",activation = enc_act, kernel_regularizer=regularizers.l2(beta), name= "encode_Y_c1")
        causal_conv2 = tf.keras.layers.Conv1D(enc_hidden//4, 2,1,padding="causal",activation = enc_act, kernel_regularizer=regularizers.l2(beta), name = "encode_Y_c2")
        signals = []
        for i in range(num_signals):
            signal = train_queue[:,:,i,:]
            signal = causal_conv(signal)
            signal = lrelu(signal)
##            signal = tf.reshape(signal, [-1,z_dim2,1,1]) # used if stride of 2 is used in encoder
##            signal = C2T1(signal)#replace causal_conv
##            signal = C2T2(signal) #replace causal_conv1
##            signal = tf.reshape(signal, [-1,signal_len,dec_hidden//2])
            signal = causal_conv1(signal)
            signal = lrelu(signal)
            signal = causal_conv2(signal)
            signal = lrelu(signal)
            signals.append(signal)
       signal = tf.concat(signals,-1)
```

```
    zm_dense = tf.keras.layers.Dense(z_dim1, kernel_regularizer=regularizers.l2(beta))
    Zm = zm_dense(signal)
     return Zm
def decode (z, reuse=False):
   with tf.variable_scope('decode') as vs:
      if reuse:
         vs.reuse_variables()
         dense = tf.keras.layers.Dense(num_signals, kernel_regularizer=regularizers.l2(beta))
          z = dense(z)
          dense1 = tf.keras.layers.Dense(num_signals, activation = term_act)
           causal_conv1 = tf.keras.layers.Conv1D(dec_hidden, 2,1,padding="causal",activation
          = dec_act, kernel_regularizer=regularizers.l2(beta))
           C2T1 = tf.keras.layers.Conv2DTranspose(dec_hidden,[2,1],[2,1],padding="same")
          C2T2 = tf.keras.layers.Conv2DTranspose(dec_hidden//2,[2,1],[2,1],padding="same")
          causal_conv2 = tf.keras.layers.Conv1D(dec_hidden//2,
          2,1,padding="causal",activation = dec_act, kernel_regularizer=regularizers.l2(beta))
          causal_conv3 = tf.keras.layers.Conv1D(dec_hidden//4,
          2,1,padding="causal",activation = dec_act, kernel_regularizer=regularizers.l2(beta))
          signals = []
          for i in range(num_signals):
                signal = causal_conv1(tf.reshape(z[:,:,i],[-1,z_dim2,1]))
                signal = causal_conv2(signal)
                signal = causal_conv3(signal)
               signals.append(signal)
        signal = tf.concat(signals,2)
```

```
        signal = dense1(signal)
        return signal

def compute_kernel(x, y):
    x_size = tf.shape(x)[0]
    y_size = tf.shape(y)[0]
    dim = tf.shape(x)[1]
    tiled_x = tf.tile(tf.reshape(x, tf.stack([x_size, 1, dim])), tf.stack([1, y_size, 1]))
    tiled_y = tf.tile(tf.reshape(y, tf.stack([1, y_size, dim])), tf.stack([x_size, 1, 1]))
    return tf.exp(-tf.reduce_mean(tf.square(tiled_x - tiled_y), axis=2) / tf.cast(dim, tf.float32))

def compute_mmd(x, y):
    x_kernel = compute_kernel(x, x)
    y_kernel = compute_kernel(y, y)
    xy_kernel = compute_kernel(x, y)
    return tf.reduce_sum(x_kernel,1) + tf.reduce_sum(y_kernel,1) - 2 * tf.reduce_sum(xy_kernel,1)#means replaced sums

train_x = tf.placeholder(tf.float32, shape=[None, signal_len,num_signals, 1])
encoded  = encode(train_x)
train_xr = decode(encoded)
#Compare the generated z with true samples from a standard Gaussian, and compute their MMD distance
loss_mmds = []
for i in range(z_dim2):#this loop used if mmd loss applied over signal dim over each time step
```

```
   true_samples = tf.random_normal(tf.stack([BATCH_SIZE, z_dim1]))# 200 instead of BATCH_SIZE

   loss_mmd_ = compute_mmd(true_samples, encoded[:,i,:])

##     loss_mmd_ =-0.5 * tf.reduce_sum(1 + Zs[:,i,:] - tf.square(Zm[:,i,:]) - tf.exp(Zs[:,i,:]), 1) VAE regularisation loss

   loss_mmds.append(loss_mmd_[:,tf.newaxis])

loss_mmd = tf.reduce_sum(tf.concat(loss_mmds,-1),-1)

#following two commented out lines used if mmd applied over all time steps and signal dim together

##true_samples = tf.random_normal(tf.stack([BATCH_SIZE, z_dim2*z_dim1]))

##loss_mmd = compute_mmd(true_samples, tf.reshape(encoded,[-1,z_dim2*z_dim1]) )

loss_nll = tf.reduce_mean(tf.reduce_mean(tf.reduce_mean(tf.square(train_xr[:,:,tf.newaxis] - train_x),-1),-1),-1)

loss = loss_nll + weight_mmd*loss_mmd

trainer = tf.train.AdamOptimizer(lr_rate).minimize(loss)


# Following code is for loading training and test data

def fillin_zeros (features1): # this function add zero rows take make all circuit data to have 12 rows

   k=0;

   features = np.zeros((n_samples*sample_size, 8011))

   for i in range(0,n_samples*sample_size,12):

      features[i+0:i+3,:] = features1[k+0:k+3,:]

      k=k+3

   return features
```

```
mat = spio.loadmat('Eparallel_input.mat', squeeze_me=True)

input_parallel1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Eseries_input.mat', squeeze_me=True)

input_series1 = mat['input']

mat = spio.loadmat('Vs_parallel_capacitor.mat', squeeze_me=True)

Vs_parallel_cap1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Vs_parallel_inductor.mat', squeeze_me=True)

Vs_parallel_ind1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Vs_R_L_RC.mat', squeeze_me=True)

Vs_R_L_RC1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Vs_R_C_RL.mat', squeeze_me=True)

Vs_R_C_RL1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Vs_RL_RC.mat', squeeze_me=True)

Vs_RL_RC1 = mat['input']

mat = spio.loadmat('Vs_R_RLC.mat', squeeze_me=True)

Vs_R_RLC1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Vs_RC_RL.mat', squeeze_me=True)

Vs_RC_RL1 = mat['input']

mat = spio.loadmat('Vs_C_LR_R.mat', squeeze_me=True)

Vs_C_LR_R1 = mat['input']

mat = spio.loadmat('Vs_L_CR_R.mat', squeeze_me=True)

Vs_L_CR_R1 = mat['input']

mat = spio.loadmat('Vs_R_LR_C.mat', squeeze_me=True)

Vs_R_LR_C1 = mat['input']
```

```
mat = spio.loadmat('Vs_R_CR_L.mat', squeeze_me=True)

Vs_R_CR_L1 = mat['input']

mat = spio.loadmat('Vs_R_L_C_R.mat', squeeze_me=True)

Vs_R_L_C_R1 = mat['input']

mat = spio.loadmat('Vs_R_C_L_R.mat', squeeze_me=True)

Vs_R_C_L_R1 = mat['input']

mat = spio.loadmat('Is_parallel_capacitor.mat', squeeze_me=True)

Is_parallel_cap1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Is_parallel_inductor.mat', squeeze_me=True)

Is_parallel_ind1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Is_R_L_RC.mat', squeeze_me=True)

Is_R_L_RC1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Is_R_C_RL.mat', squeeze_me=True)

Is_R_C_RL1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Is_RL_RC.mat', squeeze_me=True)

Is_RL_RC1 = mat['input']

mat = spio.loadmat('Is_R_RLC.mat', squeeze_me=True)

Is_R_RLC1 = fillin_zeros(mat['input'])

mat = spio.loadmat('Is_RC_RL.mat', squeeze_me=True)

Is_RC_RL1 = mat['input']

mat = spio.loadmat('Is_C_LR_R.mat', squeeze_me=True)

Is_C_LR_R1 = mat['input']

mat = spio.loadmat('Is_L_CR_R.mat', squeeze_me=True)

Is_L_CR_R1 = mat['input']
```

```
mat = spio.loadmat('Is_R_LR_C.mat', squeeze_me=True)

Is_R_LR_C1 = mat['input']

mat = spio.loadmat('Is_R_CR_L.mat', squeeze_me=True)

Is_R_CR_L1 = mat['input']

mat = spio.loadmat('Is_R_L_C_R.mat', squeeze_me=True)

Is_R_L_C_R1 = mat['input']

mat = spio.loadmat('Is_R_C_L_R.mat', squeeze_me=True)

Is_R_C_L_R1 = mat['input']


def load_circ_data(source):

    rand_circ = np.zeros((total_circuits*6*sample_size,signal_len*num_signals))

    for i in range(total_circuits):

        for j in range(6):

            mat = spio.loadmat('D:/phd/ML abstraction/RCGAN/rand_circ_4_comp_full/rand'+str(i+1)+'circuit'+str(j+1)+source+'.mat', squeeze_me=True)

            rand_circ[i*6*sample_size+j*12:i*6*sample_size+(j+1)*sample_size,:] = mat["input"][:,3:]

    return rand_circ


def row_normalisation(data_circ):

    circ_max = data_circ.max(axis=norm_axis)#-1)

    circ_min = data_circ.min(axis=norm_axis)#-1)

    circ_ranges = circ_max - circ_min

    circ_ranges[circ_ranges==0]=1
```

```
        circ_min = circ_min.reshape((-1,1))

        circ_ranges = circ_ranges.reshape((-1,1))

        data_circ = norm_scaler*np.divide(np.subtract(data_circ,circ_min),np.absolute(circ_ranges) )-norm_shifter # min max normalisation

        return data_circ


def circ_normalisation1(data_circ, num_circ): # normalisation applied over individual configuration

  for i in range(num_circ):

         for j in range(6):

           data_circ_max = np.amax(data_circ[i*6*sample_size+j*12:i*6*sample_size+(j+1)*sample_size,:] )

           data_circ_min = np.amin(data_circ[i*6*sample_size+j*12:i*6*sample_size+(j+1)*sample_size,:] )

           data_circ_ranges = data_circ_max - data_circ_min

           data_circ[i*6*sample_size+j*12:i*6*sample_size+(j+1)*sample_size,:] = norm_scaler*np.divide(np.subtract(data_circ[i*6*sample_size+j*12:i*6*sample_size+(j+1)*sample_size,:] ,data_circ_min),np.absolute(data_circ_ranges) )-norm_shifter # min max normalisation

      return data_circ


def circ_normalisation2(data_circ, num_circ): # normalisation applied over all configurations

  for i in range(num_circ):

         data_circ_max = np.amax(data_circ[i*6*sample_size:(i+1)*6*sample_size,:] )

         data_circ_min = np.amin(data_circ[i*6*sample_size:(i+1)*6*sample_size,:] )
```

```
        data_circ_ranges = data_circ_max - data_circ_min

        data_circ[i*6*sample_size:(i+1)*6*sample_size,:]                                    =
norm_scaler*np.divide(np.subtract(data_circ[i*6*sample_size:(i+1)*6*sample_size,:]
,data_circ_min),np.absolute(data_circ_ranges)  )-norm_shifter # min max normalisation

    return data_circ


rand_circ_vs = load_circ_data('Vs')#, rand_mean_vs, rand_std_vs


rand_circ_is = load_circ_data('Is')#, rand_mean_is, rand_std_is

print(rand_circ_vs.shape, "rand_circ_vs")



input1 = np.vstack(( input_series1,input_parallel1,  Vs_parallel_cap1,  Vs_parallel_ind1,
Vs_R_L_RC1, Vs_R_C_RL1, Vs_R_RLC1, Is_parallel_ind1, Is_parallel_cap1, Is_R_L_RC1,
Is_R_C_RL1, Is_R_RLC1,\

            Is_RL_RC1, Is_RC_RL1, Vs_RL_RC1,  Vs_RC_RL1,\

            Vs_R_LR_C1,  Vs_R_C_L_R1,  Vs_L_CR_R1,  Vs_R_CR_L1,  Vs_C_LR_R1,
Vs_R_L_C_R1, Is_R_C_L_R1, Is_R_L_C_R1, Is_C_LR_R1,  Is_R_CR_L1, Is_L_CR_R1,
Is_R_LR_C1))


input1=                                                input1[:,3:].reshape((-1,num_signals+3,
signal_len+1))[:,:num_signals,:signal_len].reshape((-1,num_signals*signal_len))

all_data = np.vstack((input1, rand_circ_vs, rand_circ_is))

all_data = circ_normalisation1(all_data, 2*total_circuits+(total_sets))

X_train = all_data[input1.shape[0]:]

X_val = all_data[:input1.shape[0]]
```

```
input1 = []

all_data = []

rand_circ_vs = []

rand_circ_is = []


def pernode_normal(features, curr_indx=None):

   features = features[:,3:].reshape((-1,num_signals+3, signal_len+1))[:,:num_signals,:signal_len].reshape((-1,num_signals*signal_len))

  features = circ_normalisation1(features,1)

  return features


def zero_extend(features1):

   features1 = pernode_normal(features1)

   features1 =features1.flatten()

   features1 = features1.reshape((1,features1.shape[0]))

   return features1


input_series1 = pernode_normal(input_series1)

input_series =input_series1.flatten()

input_series = input_series.reshape((1,input_series.shape[0]))

Vs_RL_RC1 = pernode_normal(Vs_RL_RC1)

Vs_RL_RC =Vs_RL_RC1.flatten()

Vs_RL_RC = Vs_RL_RC.reshape((1,Vs_RL_RC.shape[0]))

Is_RL_RC1 = pernode_normal(Is_RL_RC1)
```

```
Is_RL_RC =Is_RL_RC1.flatten()

Is_RL_RC = Is_RL_RC.reshape((1,Is_RL_RC.shape[0]))

Vs_RC_RL1 = pernode_normal(Vs_RC_RL1)

Vs_RC_RL =Vs_RC_RL1.flatten()

Vs_RC_RL = Vs_RC_RL.reshape((1,Vs_RC_RL.shape[0]))


Is_RC_RL1 = pernode_normal(Is_RC_RL1)

Is_RC_RL =Is_RC_RL1.flatten()

Is_RC_RL = Is_RC_RL.reshape((1,Is_RC_RL.shape[0]))

Vs_C_LR_R1 = pernode_normal(Vs_C_LR_R1)

Vs_C_LR_R =Vs_C_LR_R1.flatten()

Vs_C_LR_R = Vs_C_LR_R.reshape((1,Vs_C_LR_R.shape[0]))

Is_C_LR_R1 = pernode_normal(Is_C_LR_R1)

Is_C_LR_R =Is_C_LR_R1.flatten()

Is_C_LR_R = Is_C_LR_R.reshape((1,Is_C_LR_R.shape[0]))

Vs_L_CR_R1 = pernode_normal(Vs_L_CR_R1)

Vs_L_CR_R =Vs_L_CR_R1.flatten()

Vs_L_CR_R = Vs_L_CR_R.reshape((1,Vs_L_CR_R.shape[0]))

Is_L_CR_R1 = pernode_normal(Is_L_CR_R1)

Is_L_CR_R =Is_L_CR_R1.flatten()

Is_L_CR_R = Is_L_CR_R.reshape((1,Is_L_CR_R.shape[0]))

Vs_R_CR_L1 = pernode_normal(Vs_R_CR_L1)

Vs_R_CR_L =Vs_R_CR_L1.flatten()

Vs_R_CR_L = Vs_R_CR_L.reshape((1,Vs_R_CR_L.shape[0]))
```

```
Is_R_CR_L1 = pernode_normal(Is_R_CR_L1)

Is_R_CR_L =Is_R_CR_L1.flatten()

Is_R_CR_L = Is_R_CR_L.reshape((1,Is_R_CR_L.shape[0]))

Vs_R_LR_C1 = pernode_normal(Vs_R_LR_C1)

Vs_R_LR_C =Vs_R_LR_C1.flatten()

Vs_R_LR_C = Vs_R_LR_C.reshape((1,Vs_R_LR_C.shape[0]))

Is_R_LR_C1 = pernode_normal(Is_R_LR_C1)

Is_R_LR_C =Is_R_LR_C1.flatten()

Is_R_LR_C = Is_R_LR_C.reshape((1,Is_R_LR_C.shape[0]))

Vs_R_L_C_R1 = pernode_normal(Vs_R_L_C_R1)

Vs_R_L_C_R =Vs_R_L_C_R1.flatten()

Vs_R_L_C_R = Vs_R_L_C_R.reshape((1,Vs_R_L_C_R.shape[0]))

Is_R_L_C_R1 = pernode_normal(Is_R_L_C_R1)

Is_R_L_C_R =Is_R_L_C_R1.flatten()

Is_R_L_C_R = Is_R_L_C_R.reshape((1,Is_R_L_C_R.shape[0]))

Vs_R_C_L_R1 = pernode_normal(Vs_R_C_L_R1)

Vs_R_C_L_R =Vs_R_C_L_R1.flatten()

Vs_R_C_L_R = Vs_R_C_L_R.reshape((1,Vs_R_C_L_R.shape[0]))

Is_R_C_L_R1 = pernode_normal(Is_R_C_L_R1)

Is_R_C_L_R =Is_R_C_L_R1.flatten()

Is_R_C_L_R = Is_R_C_L_R.reshape((1,Is_R_C_L_R.shape[0]))

input_parallel = zero_extend(input_parallel1)

Vs_parallel_cap = zero_extend(Vs_parallel_cap1)

Is_parallel_cap = zero_extend(Is_parallel_cap1)
```

```
Vs_parallel_ind = zero_extend(Vs_parallel_ind1)

Is_parallel_ind = zero_extend(Is_parallel_ind1)

Vs_R_L_RC = zero_extend(Vs_R_L_RC1)

Is_R_L_RC = zero_extend(Is_R_L_RC1)

Vs_R_C_RL = zero_extend(Vs_R_C_RL1)

Is_R_C_RL = zero_extend(Is_R_C_RL1)

Vs_R_RLC = zero_extend(Vs_R_RLC1)

Is_R_RLC = zero_extend(Is_R_RLC1)

plt.ion()

batch_size = BATCH_SIZE

sess = tf.Session()

saver = tf.train.Saver()

if recall_save == True:

    saver = tf.train.import_meta_graph('embeddings_single_row/my_model.meta')

    #comment out run

    saver.restore(sess, tf.train.latest_checkpoint('embeddings_single_row/'))

else:

    sess.run(tf.global_variables_initializer())

j=0

# Start training

for k in range(training_epochs):

    nll = 0.

    mmd = 0.

    nll_2 = 0.
```

```
    batch_xs = np.zeros((BATCH_SIZE,signal_len,num_signals,1))
    batch_vs = np.zeros((BATCH_SIZE,signal_len,num_signals,1))
    count =0
    while count < BATCH_SIZE-1:
        idx = np.random.choice(np.arange(0, X_train.shape[0]))#, 12))
        if np.count_nonzero(X_train[idx]) >0:
            batch_xs[count] = X_train[idx,:num_signals*signal_len][np.newaxis,...].reshape(-1,signal_len,num_signals,1)
            if count == (BATCH_SIZE - 1):
                break
            count +=1
    count =0
    while count < BATCH_SIZE-1:
        idx = np.random.choice(np.arange(0, X_val.shape[0]))#, 12))
        if np.count_nonzero(X_val[idx]) >0:
            batch_vs[count] = X_val[idx,:num_signals*signal_len][np.newaxis,...].reshape(-1,signal_len,num_signals,1)
            if count == (BATCH_SIZE - 1):
                break
            count +=1
    # Fit training using batch data
    _, nll,mmd = sess.run([trainer, loss_nll, loss_mmd], feed_dict={train_x: batch_xs })
    nll_v,mmd_v = sess.run([loss_nll, loss_mmd], feed_dict={train_x: batch_vs })
    print("epoch %i train nll is %f, train mmd is %f, val nll is %f, val mmd is %f" % (k, np.mean(nll), np.mean(mmd), np.mean(nll_v), np.mean(mmd_v)))
```

```
    if k >0 and k % save_epochs == 0:
        if math.isnan(np.mean(nll))==False :
            save_path = saver.save(sess, "embeddings_single_row/my_model")
            print("vars saved in file: %s" % save_path)

sample1 = sess.run(train_xr, feed_dict={train_x: input_parallel[:,0:num_signals*signal_len].reshape(-1, signal_len,num_signals, 1)}

samples = sess.run(train_xr, feed_dict={train_x: input_series[:,0:num_signals*signal_len].reshape(-1,signal_len,num_signals,  1)})

np.savetxt("predicted.csv",  samples.reshape(-1,1*num_signals*signal_len), delimiter=",")

fig_serie, axs = plt.subplots(3, 1)

axs[0].plot(np.arange(signal_len), input_series[0,3*signal_len:4*signal_len])

axs[0].plot(np.arange(signal_len), samples.reshape(1,num_signals*signal_len)[0,3*signal_len:4*signal_len])

axs[1].plot(np.arange(signal_len), input_series[0,4*signal_len:5*signal_len])

axs[1].plot(np.arange(signal_len), samples.reshape(1,num_signals*signal_len)[0,4*signal_len:5*signal_len])

axs[2].plot(np.arange(signal_len), input_parallel[0,4*signal_len:num_signals*signal_len])

axs[2].plot(np.arange(signal_len), sample1.reshape(1, num_signals*signal_len)[0,4*signal_len:num_signals*signal_len])

fig_serie.show()

sample2 = sess.run(train_xr, feed_dict={train_x: Vs_R_RLC[:,0:num_signals*signal_len].reshape(-1,signal_len,num_signals,  1)})

fig_serie2, axs2 = plt.subplots(4, 1)
```

```
axs2[0].plot(np.arange(signal_len), Vs_R_RLC[0,0*signal_len:1*signal_len])

axs2[0].plot(np.arange(signal_len), sample2.reshape(1, num_signals*signal_len)[0,0*signal_len:1*signal_len])

axs2[1].plot(np.arange(signal_len), Vs_R_RLC[0,1*signal_len:2*signal_len])

axs2[1].plot(np.arange(signal_len), sample2.reshape(1, num_signals*signal_len)[0,1*signal_len:2*signal_len])

axs2[2].plot(np.arange(signal_len), Vs_R_RLC[0,2*signal_len:3*signal_len])

axs2[2].plot(np.arange(signal_len), sample2.reshape(1, num_signals*signal_len)[0,2*signal_len:3*signal_len])

axs2[3].plot(np.arange(signal_len), Vs_R_RLC[0,4*signal_len:5*signal_len])

axs2[3].plot(np.arange(signal_len), sample2.reshape(1, num_signals*signal_len)[0,4*signal_len:5*signal_len])

fig_serie2.show()

z_mu = np.zeros((n_samples*sample_size,z_dim1*z_dim2,total_sets))

batch_xs1 = np.zeros((BATCH_SIZE,trial_size))

for select in range(1,total_sets):

   for i in range(0,n_samples*sample_size):#n_samples

      if select ==1:

         batch_xs =  Vs_R_L_RC[:,trial_size*i:trial_size*(i+1)]

      if select ==2:

         batch_xs = input_parallel[:,trial_size*i:trial_size*(i+1)]

      if select ==3:

         batch_xs =  Vs_parallel_cap[:,trial_size*i:trial_size*(i+1)]

      if select ==4:

         batch_xs = Vs_parallel_ind[:,trial_size*i:trial_size*(i+1)]

      if select ==5:
```

```
    batch_xs =  Vs_R_C_RL[:,trial_size*i:trial_size*(i+1)]
if select ==6:
    batch_xs = Vs_R_RLC[:,trial_size*i:trial_size*(i+1)]
if select ==7:
    batch_xs =  Is_parallel_cap[:,trial_size*i:trial_size*(i+1)]
if select ==8:
    batch_xs = Is_parallel_ind[:,trial_size*i:trial_size*(i+1)]
if select ==9:
    batch_xs =  Is_R_C_RL[:,trial_size*i:trial_size*(i+1)]
if select ==10:
    batch_xs = Is_R_RLC[:,trial_size*i:trial_size*(i+1)]
if select ==11:
    batch_xs = Is_R_L_RC[:,trial_size*i:trial_size*(i+1)]
if select ==12:
    batch_xs = Vs_RL_RC[:,trial_size*i:trial_size*(i+1)]
if select ==13:
    batch_xs = Vs_RC_RL[:,trial_size*i:trial_size*(i+1)]
if select ==14:
    batch_xs = Is_RL_RC[:,trial_size*i:trial_size*(i+1)]
if select ==15:
    batch_xs = Is_RC_RL[:,trial_size*i:trial_size*(i+1)]
if select ==16:
    batch_xs = Vs_C_LR_R[:,trial_size*i:trial_size*(i+1)]
if select ==17:
```

```
        batch_xs = Is_C_LR_R[:,trial_size*i:trial_size*(i+1)]
    if select ==18:
        batch_xs = Vs_L_CR_R[:,trial_size*i:trial_size*(i+1)]
    if select ==19:
        batch_xs = Is_L_CR_R[:,trial_size*i:trial_size*(i+1)]
    if select ==20:
        batch_xs = Vs_R_CR_L[:,trial_size*i:trial_size*(i+1)]
    if select ==21:
        batch_xs = Is_R_CR_L[:,trial_size*i:trial_size*(i+1)]
    if select ==22:
        batch_xs = Vs_R_LR_C[:,trial_size*i:trial_size*(i+1)]
    if select ==23:
        batch_xs = Is_R_LR_C[:,trial_size*i:trial_size*(i+1)]
    if select ==24:
        batch_xs = Vs_R_C_L_R[:,trial_size*i:trial_size*(i+1)]
    if select ==25:
        batch_xs = Is_R_C_L_R[:,trial_size*i:trial_size*(i+1)]
    if select ==26:
        batch_xs = Vs_R_L_C_R[:,trial_size*i:trial_size*(i+1)]
    if select ==27:
        batch_xs = Is_R_L_C_R[:,trial_size*i:trial_size*(i+1)]
    batch_xs1[i%BATCH_SIZE,:]= batch_xs
    if (i+1)%BATCH_SIZE==0:# and np.sum(batch_xs1)>0:
```

```
        z_mu[i-(BATCH_SIZE-1):(i+1),:,select-1] = np.reshape(sess.run(encoded, feed_dict={train_x: batch_xs1.reshape(-1,signal_len,num_signals, 1)}),[-1,z_dim2*z_dim1])
    direc = "embeddings_single_row/z_mu"+str(select)+"_full.csv";
    np.savetxt(direc, z_mu[:,:,select-1], delimiter=",");


z_mu_test = np.zeros((n_samples*sample_size,z_dim1*z_dim2))# series RLC is embedded separately as initially it was being used as test data
for k in range(0,n_samples*sample_size,BATCH_SIZE):
    z_mu_test[k:k+BATCH_SIZE,:] = np.reshape(sess.run(encoded, feed_dict={train_x: input_series[:,trial_size*k : trial_size*(k+BATCH_SIZE)].reshape(-1,signal_len,num_signals, 1)}), [-1,z_dim2*z_dim1])#trial_size*(j-1):trial_size*(j+(BATCH_SIZE-1))])
direc = "embeddings_single_row\z_mu_test_full.csv";
np.savetxt(direc, z_mu_test, delimiter=",");#select,i,


z_mu1 = np.zeros(((total_sets)*n_samples*sample_size,z_dim1*z_dim2))# reshaping for isomap embeddings
for n in range(total_sets):
    for i in range(0,n_samples*sample_size):
        z_mu1[(n_samples*sample_size)*n+i,:] = z_mu[i,:,n]


iso = manifold.Isomap(n_neighbors=24, n_components=ISOMAP_dim)
iso.fit(z_mu1)
manifold_2Da = iso.transform(z_mu1)
np.savetxt("z_ISOMAP_1row_24neighb.csv", manifold_2Da, delimiter=",")
#similarly isomap embedding for 12 and 72 neighbours can be generated
```